QUICK START SERIES

سیکھئے

محمد ذوالقرنین چوہدری

JBD Press

QUICK START SERIES

کچھ کتاب کے بارے میں

- باب نمبر 1
اس میں ++C کی تاریخ اور جن عوامل کو مدنظر رکھتے ہوئے یہ لینگوئج بنائی گئی ہے اس سے متعلق تمام معلومات شامل ہیں۔
- باب نمبر 2
میں ++C کنٹرول سٹیٹ، سینٹ لوپس اور فنکشنز شامل ہیں۔ یہ تینوں چیزیں ہر لینگوئج کی جان ہوتی ہیں
- باب نمبر 3
میں آپ اریز اور سٹرنگ کے بارے میں تفصیل سے پڑھیں گے۔
- باب نمبر 4
پوائنٹر سے متعلق ہے۔ پوائنٹر زC میں پہلی دفعہ شامل کیے گئے کیونکہ ++C اصل میں C سے derived فارم ہے اس لئے یہ اس میں بھی شامل کیے گئے ہیں۔
- باب نمبر 5
میں آپ اوبجیکٹ اوریئنٹڈ پروگرامنگ میں کام کرنا شروع کریں گے۔ یعنی یہ سٹرکچر اور کلاسز سے متعلق آپ کو معلومات فراہم کرے گا۔
- باب نمبر 6
ایک اضافی فیچر کے بارے میں ہے جس میں آپ ہر آپریٹر سے اپنی مرضی کا آپریشن پرفام کرا سکتے ہیں۔
- باب نمبر 7
++C کی جان ہے اس میں آپ پیور پروگرامنگ کے بارے میں پڑھیں گے۔

اُردو بازار لاہور۔ فون: 042-7220879
اقبال روڈ راولپنڈی۔ فون: 051-5539609
اُردو بازار کراچی۔ فون: 021-7765086

ہماری چند کمپیوٹر کتب

QUICK START SERIES

سیکھئے

محمد ذوالقرنین چوہدری

جہانگیر بُک ڈپو
لاہور۔راولپنڈی۔کراچی

انتساب

چوہدری محمد اویس کے نام

جن کی اعانت اور رہنمائی سے

میں نے یہ مقام حاصل کیا

حرفِ اوّل

ہم آپ کو پیور اوبجیکٹ اوورنٹیڈ پروگرامنگ لینگویج میں خوش آمدید کہتے ہیں۔ یہ کتاب خاص طور پر اوبجیکٹ اوورنٹیڈ پروگرامنگ کی بنیادی معلومات سیکھنے کے لئے ترتیب دی گئی ہے۔ گزشتہ چند سالوں سے ہارڈویئر میں بہت تیزی سے ترقی ہورہی ہے لیکن کچھ ناگزیر وجوہات کی بنا پر سافٹ ویئر میں خاطر خواہ ترقی نہیں ہورہی۔ لوگوں کا سیکھنے کا رجحان ہارڈویئر کی طرف زیادہ ہے کیونکہ سافٹ ویئر کی افادیت کا انہیں اندازہ نہیں اور نہ ہی ان کو پروگرامنگ کے بارے میں بنیادی معلومات تفصیلاً فراہم کی جاتی ہیں۔ اس بات کو پیش نظر رکھتے ہوئے میں نے اوبجیکٹ اوورنٹیڈ لینگویج کے بارے میں ایک کتاب ترتیب دی ہے۔ اس میں آپ کو اوبجیکٹ اوورنٹیڈ پروگرامنگ کی افادیت اور اس کی بنیادی معلومات تفصیلاً فراہم کرنے کی کوشش کی گئی ہے۔ میں امید کرتا ہوں کہ یہ آپ کو پسند آئے گی۔

اس وقت دنیا میں سب سے زیادہ استعمال ہونے والی لینگویج اوبجیکٹ اوورنٹیڈ ہے لیکن بدقسمتی سے پاکستان میں ++C میں اتنا زیادہ کام نہیں ہورہا ہے۔ اس کی وجہ یہ ہے کہ نئی نسل کے اس لینگویج کے بارے میں کانسیپٹس کلیئر نہیں ہیں۔ اس کو ذہن میں رکھتے ہوئے میں نے اس کتاب میں ++C کے بنیادی کانسیپٹس سادہ اور آسان زبان میں سمجھانے کی کوشش کی ہے اور مجھے پورا یقین ہے کہ آپ تھوڑی سی محنت کے بعد بغیر کسی ٹیوٹر کے اس کتاب کی مدد سے ++C لینگویج خود سے سیکھ لیں گے۔ اس کتاب میں کل سات ابواب ہیں اور ہر موضوع کو کم از کم ایک مثال کی مدد سے سمجھایا گیا ہے۔ اس کے علاوہ ہر باب کے آخر پر مشق دی گئی ہے۔ جس میں روزمرہ زندگی سے متعلقہ سوالات اور بعد میں ان کے جوابات بھی تحریر کئے گئے ہیں۔ ایک اہم بات جس کا میں یہاں تذکرہ ضروری سمجھتا ہوں کہ آپ اس لینگویج کی جتنی زیادہ مشق یعنی پریکٹس کریں گے اس کے بارے میں آپ کے کانسیپٹس اتنے ہی واضح ہوں گے اور مشہور کہاوت بھی ہے کہ "Practice makes a man perfect" میں امید کرتا ہوں کہ آپ کو یہ کتاب پسند آئے گی۔ آپ کے خیال میں اس میں اگر کمی وبیشی کی گنجائش ہے تو براہ مہربانی مجھے ضرور اطلاع دیجئے گا۔ میں آپ کا بے حد ممنون رہوں گا۔

چوہدری محمد ذوالقرنین
M.C.S, B.C.S, E-Commerce
zaki352@hotmail.com

گزارش

اس کتاب کو ترتیب دینے میں کئی لوگوں اور میرے اساتذہ کی معاونت مجھے درکار رہی ہے۔ ان میں سے بعض لوگوں نے پروگرامنگ سے متعلق اپنے تجربہ کی روشنی میں میری رہنمائی کی۔ دوسروں نے اس ایڈیشن کا مطالعہ کیا اور اس کتاب کے مواد اور پروگرامنگ لینگویج کے کانسپٹس (Concepts) کو موثر طریقے سے بیان کرنے کے بارے میں اپنی مفید تجاویز سے نوازا۔ میں اس سلسلہ میں خاص طور پر اپنے بڑے بھائی چوہدری محمد اویس صاحب اور عدیل نیاز صاحب کا خاص ممنون ہوں۔

ان کے علاوہ میں ان احباب کا بھی مشکور ہوں جو گاہے بگاہے اپنی مفید مشاورت سے مجھے نوازتے رہے۔ بالخصوص میں ان احباب کا بے حد ممنون ہوں۔

- میاں محمد اشرف صاحب(Director CIMIT)
- محمد اختر خان صاحب (میک انفارمیٹکس گوجرہ)
- پروفیسر سید کامران زیدی
- شقران احمد خان
- رضوان اکرم
- مس سیما کنول
- چوہدری محمد شعیب
- ظہیر احمد قریشی صاحب(Director C.B.A)
- پروفیسر عبدالخالق صاحب(M.I.S)
- اکرام الحق
- یاسر اورنگ زیب
- محمد الطاف حسین
- سید عظیم شاہ(M.C.S, OCP, M.C.S.D)
- محمد اکمل شہزاد(Computer Operator)

چوہدری محمد ذوالقرنین

M.C.S, B.C.S, E-Commerce

فہرست

"کتاب کے بارے میں"

شروع اللہ کے نام سے جو بڑا مہربان ہے۔

C++ لینگویج کی اہمیت کو مدنظر رکھتے ہوئے میں نے کوشش کی ہے کہ اسی کتاب میں آپ کو C++ کی تمام بنیادی معلومات فراہم کی جائیں جو ایک پروگرامر کے لئے ضروری ہوتی ہیں۔ دوسری کتابوں کی نسبت یہ کتاب بہت معیاری ہے۔ اس لئے کہ اس میں ہر بات ٹو دی پوائنٹ کی گئی ہے اور ہر عنوان یا C++ کمانڈ ایک حقیقی زندگی پر مشتمل مثال کی مدد سے سمجھانے کی کوشش کی گئی ہے۔ اس کے علاوہ اس کتاب کا مواد بہت آسان رکھا گیا ہے تا کہ ہر خاص و عام کی سمجھ میں آ جائے۔ اس کتاب کے کل سات باب ہیں۔ آیئے سب کو ایک نظر دیکھتے ہیں کہ ان میں کیا ہے؟

باب نمبر 1:

اس میں C++ کی تاریخ اور جن عوامل کو مدنظر رکھتے ہوئے یہ لینگویج بنائی گئی۔ اس سے متعلقہ تمام معلومات شامل ہے۔ اس کے علاوہ اس میں C++ کا بنیادی انٹرفیس بتایا گیا ہے کہ آپ کس طرح ایک آسان سا پروگرام بنا سکتے ہیں۔

باب نمبر 2:

یہ باب بنیادی لحاظ سے کافی اہمیت کا حامل ہے کیونکہ اس میں C++ کنٹرول سٹیٹ مینٹ لوپس اور فنکشنز شامل ہیں۔ یہ تینوں چیزیں ہر لینگویج کی جان ہوتی ہیں اور ان کے بغیر آپ اپنی کوئی بھی اہم پرابل حل نہیں کر سکتے۔ اس میں آپ کو بتایا گیا کہ کب اور کیسے آپ نے فیصلہ کرنا ہے اور پروگرام کی کوڈنگ کس طرح کرنی ہے۔

باب نمبر 3:

اس باب میں آمپ اریز اور سٹرنگ کے بارے میں تفصیل سے پڑھیں گے۔ یعنی اریز میں سے کس طرح ویلیو حاصل کرتے ہیں۔ اریز کو ترتیب دینا پھر کوئی ویلیو اس کے درست آرڈر پر لکھنا اس کے علاوہ سٹرنگ ان پٹ کیسے لی جاتی ہے اور سٹرنگ کے ساتھ آپ کیسے آپریشن پرفارم کرتے ہیں۔ مثلاً دو سٹرنگز کو جمع کرنا یا کاپی کرنا وغیرہ۔

باب نمبر 4:

یہ باب پوائنٹر سے متعلق ہے۔ پوائنٹر ز C میں پہلی دفعہ شامل کئے گئے اور کیونکہ C++ اصل میں C کی ڈرائیوڈ فارم ہے اس لئے یہ اس میں بھی شامل ہیں۔ پوائنٹر ز کافی مشکل ہیں لیکن ان کا فائدہ بہت ہے ان کی مدد سے آپ سرچ کا کام دوسرے پروگرام کی نسبت بہت آسانی سے کر سکتے ہیں۔ اس کے علاوہ فنکشن آرگومینٹ Dynamic بنا سکتے ہیں۔ یہ کام آپ باب نمبر 4 میں پڑھیں گے۔

باب نمبر 5:

اس باب سے آپ اوبجیکٹ اوورنٹیڈ پروگرامنگ میں کام شروع کردیں گے۔ یعنی یہ سٹرکچر اور کلاسز سے متعلق آپ کو معلومات فراہم کرے گا۔ اس میں پروگرامنگ اسی باب سے شروع ہوتی ہے۔ اس سے پہلے صرف بنیادی معلومات تھی جو پروگرامنگ کے لئے ضروری ہوتی ہے۔

باب نمبر 6:

یہ باب ایک اضافی فیچر کے بارے میں ہے۔ جس میں آپ ہر آپریٹر سے اپنی مرضی کا آپریشن پرفام کرا سکتے ہیں یعنی اس میں آپ آپریٹر اوورلوڈنگ کریں گے۔

باب نمبر 7:

یہ ہماری کتاب کا آخری باب ہے اور ++C کی جان ہے۔ اس میں آپ پیور پروگرامنگ کے بارے میں پڑھیں گے۔ کہ پہلے سے بنی ہوئی چیز کو دوبارہ کس طرح استعمال کرتے ہیں یعنی اس میں ایک سافٹ ویئر کو بار بار استعمال کرنے کے بارے میں بتایا گیا ہے اور اس کے علاوہ آپ اپنے ڈیٹا کو مستقل سٹور کرنے کا بھی طریقہ پڑھیں گے یہ باب انہیر ٹینس (Inheritance)، پولی مارفزیم اور فائل پروسیسنگ سے متعلقہ ہے۔
اس کے علاوہ ہم نے اس کتاب کے ہر باب کے آخر پر مشق لکھی ہے۔ جس میں حقیقی زندگی سے متعلقہ سوالات ہیں یعنی جو آپ روزمرہ زندگی میں ایک دوسرے سے پوچھتے ہیں اور بعد میں ان سوالات کے جوابات بھی درج ہیں۔

باب نمبر 1

C++ پروگرامنگ کا تعارف

ہم آپ کو C++ میں خوش آمدید کہتے ہیں۔ یہ C++ پروگرامنگ کا پہلا باب ہے اور اس باب میں آپ پڑھیں گے کہ C++ کیا ہے؟ اور یہ کیوں پڑھی جاتی ہے یعنی اسے پڑھنے کا کیا فائدہ ہے؟ اس باب میں آپ C++ سے متعلق بنیادی معلومات حاصل کریں گے۔ C++ بہت مشکل لینگوئج ہے اور سب سے اہم یہ ہے کہ C++ میں بڑے حروف اور چھوٹے حروف کا خیال رکھنا بہت ضروری ہے یعنی یہ ایک Case sensitive لینگوئج ہے۔ اس باب میں آپ C++ کے مندرجہ ذیل بنیادی اصول پڑھیں گے۔

- کمپیوٹر کیا ہے؟
- لینگوئجز
- C++ کی تاریخ
- C++ کی سٹینڈرڈ لائبریری
- C++ پروگرامز کا تعارف
- C++ پروگرام لکھنا
- ویری ایبلز
- کی ورڈز اور ایڈینٹیفائرز
- نیو لائن کریکٹر
- ڈیٹا ٹائپس
- int ڈیٹا ٹائپ
- کریکٹر ڈیٹا ٹائپ
- فلوٹنگ پوائنٹ ڈیٹا ٹائپس
- حسابی آپریٹرز
- آپریٹر کی فوقیت
- یونری آپریٹرز
- آرتھمیٹک آسائنمنٹ آپریٹرز
- ریلیشنل آپریٹرز
- ٹائپ کنورژن
- ان پٹ
- سٹرنگ ان پٹ
- مشق

کمپیوٹر کیا ہے؟

کمپیوٹر ایک الیکٹرانک مشین ہے جو کہ مختلف آپریشنز پرفارم کرتی ہے اس کے علاوہ یہ لاجیکل کام کرنے کے بھی اہل ہے اور یہ تمام کام ایک انسانی دماغ سے ملین یا بلین گنا تیز کرتی ہے۔ جیسا کہ آج کل کا ایک کمپیوٹر کئی سوملین ایڈیشنز ایک سیکنڈ میں پرفارم کر سکتا ہے۔ کمپیوٹر ڈیٹا کو کچھ انسٹرکشن کے مطابق کنٹرول یا پراسس کرتا ہے اور یہ ہدایات پروگرام کہلاتی ہیں۔ اس کے علاوہ کمپیوٹر کی فزیکل باڈی چھ یونٹس سے مل کر بنتی ہے اور وہ یہ ہیں۔

ان پٹ یونٹ:

یہ کمپیوٹر کا معلومات حاصل کرنے والا یونٹ ہے۔ اس میں کمپیوٹر مختلف ان پٹ ڈیوائسس سے معلومات حاصل کرتا ہے اور اس معلومات کو اس طرح منظم کرتا ہے کہ یہ معلومات بعد میں بھی پراسس کی جاسکیں۔ آج کل زیادہ معلومات کی بورڈ یا ماؤس کی مدد سے کمپیوٹر میں سٹور کی جاتی ہیں اس کے علاوہ اب آپ بول کر بھی معلومات کمپیوٹر میں سٹور کر سکتے ہیں۔

آؤٹ پٹ یونٹ:

یہ سیکشن ان پٹ یونٹ کے برعکس ہے۔ اس میں وہ معلومات جو پراسس کی جا چکی ہوں وہ آؤٹ پٹ یونٹ پر ظاہر کی جاتی ہیں تاکہ یوزر اسے دیکھ سکے اور اگر وہ کمپیوٹر کے باہر بھی اسے نکالنا چاہتا ہے تو نکال سکے۔ مثلاً مانیٹر یا پرنٹر اس میں اہم کردار ادا کرتے ہیں۔

میموری یونٹ:

یہ کمپیوٹر کا عارضی دماغ ہوتا ہے۔ آپ ان پٹ یونٹ کی مدد سے جو معلومات کمپیوٹر کو فراہم کرتے ہیں وہ یہ معلومات میموری میں محفوظ کر لیتا ہے تاکہ بعد میں جب بھی اس کو پراسس کرنے کی ضرورت پیش آئے تو آپ کو دوبارہ یہ معلومات فراہم نہ کرنی پڑیں بلکہ کمپیوٹر سے ہی حاصل کر لیں اور اس کے علاوہ جب تک آپ پراسس کی ہوئی انفارمیشن کو آؤٹ پٹ یونٹ پر نہیں لے جاتے وہ بھی اس یونٹ میں محفوظ رہتی ہے۔

آرتھمیٹک اینڈ لاجک یونٹ (ALU):

یہ وہ یونٹ ہے جس میں آپ کا کمپیوٹر ڈیٹا کو پراسس کرتا ہے۔ اس میں کیلکولیشن مثلاً ایڈیشن یا تقسیم وغیرہ کی جاتی ہے۔ اس میں ایک سسٹم ہوتا ہے جو کہ خود سے فیصلہ کر سکتا ہے۔ مثلاً دونمبروں کا موازنہ کرنا ہے کہ کیا وہ برابر ہیں یا نہیں وغیرہ۔

سینٹرل پراسیسنگ یونٹ (CPU):

یہ تمام کمپیوٹر کا ہیڈ ہے اور یہ دوسرے سیکشنز کی نگرانی کرتا ہے کہ انہوں نے کون کون سے آپریشنز پرفارم کئے ہیں۔ یہ ان پٹ یونٹ کو بتاتا ہے کہ کب میموری یونٹ سے معلومات حاصل کرنی ہے اور کب ALU کو اسے پراسس کرنا ہے۔ غرض کہ یہ تمام دوسرے یونٹس کو آرڈر دیتا ہے کہ فلاں کام کرو۔

سیکنڈری سٹوریج یونٹ:

یہ میموری یونٹ سے کہیں بڑا ہوتا ہے اور یہ کمپیوٹر کا مستقل دماغ ہے جس میں آپ تمام ڈیٹا خواہ وہ پراسس ہو چکا ہے یا نہیں، محفوظ کر سکتے ہیں۔ مثلاً ہارڈ ڈسک پر ڈیٹا سٹور کرنا وغیرہ۔

آپ نے اوپر کمپیوٹر کے بارے میں پڑھا ہے کہ وہ کس طرح کام کرتا ہے۔ آپ نے اس کے علاوہ Personal computer کا لفظ بھی سنا ہوگا یہ 1977ء میں ایپل کمپیوٹر والوں نے متعارف کروایا تھا۔ اس سے کمپیوٹر بہت سستا ہو گیا اور اب اسے لوگ آسانی سے اپنے ذاتی کام کے لئے خرید سکتے تھے۔ پھر 1981ء میں IBM نے پرسنل کمپیوٹر متعارف کروایا۔

لینگویجز:

ہماری یہ کتاب ++C سے متعلقہ ہے اور میرا خیال ہے کہ آپ کو اس سے پہلے عام لینگویجز کے بارے میں بھی معلومات حاصل ہونی چاہئے کہ ++C کو ان پر فوقیت کیوں دی جاتی ہے۔ پروگرامرز مختلف لینگویجز میں کمپیوٹر کے لئے پروگرام لکھتے ہیں۔ ان میں سے کچھ لینگویجز میں لکھا ہوا کوڈ یا پروگرام کمپیوٹر سمجھ لیتا ہے لیکن بعض کو سمجھنے کے لئے اسے کسی ٹرانسلیٹر کی ضرورت ہوتی ہے۔ بالکل ایسے ہی جیسے آپ کسی ملک میں جائیں جہاں کی زبان آپ کو نہ آتی ہو تو آپ کو ایک مترجم کی ضرورت ہوگی جو آپ کو ان لوگوں کی اور ان لوگوں کو آپ کی زبان سمجھائے گا۔ آجکل بہت زیادہ کمپیوٹر لینگویجز استعمال ہو رہی ہیں۔ ان کو تین حصوں میں تقسیم کیا گیا ہے۔

Machine Language

Assembly Language

High-level Language

مشین لینگویج نام سے ظاہر ہے کہ یہ کمپیوٹر کی مخصوص زبان ہی ہوگی اور ایسا ہے بھی کیونکہ کمپیوٹر صرف اس لینگویج کو ڈائریکٹ بغیر کسی ٹرانسلیٹر کے سمجھتا ہے اس میں کوڈ صرف 0's اور 1's میں لکھا جاتا ہے۔ یہ لینگویجز مشین پر انحصار کرتی ہیں کیونکہ یہ مشین کے ہارڈویئر کے مطابق ڈیفائن کی جاتی ہیں۔ یہ لینگویجز انسانی دماغ کے لئے سمجھنا بہت مشکل ہیں۔

جیسے جیسے وقت گزرتا گیا کمپیوٹر میں بھی ترقی ہوتی گئی اور پھر زیادہ تر پروگرامرز نے یہ محسوس کیا کہ مشین لینگویج بہت سست اور مشکل ہے۔ اس لئے پروگرامرز نے سٹرنگ نمبرز کی بجائے انگلش کے حروف استعمال کرنا شروع کر دیئے۔ جن کی بنیاد پر پھر اسمبلی لینگویج بنائی گئی۔ پھر بعد میں اس لینگویج کو کمپیوٹر کے لئے آسان اور تیز بنانے کے لئے ٹرانسلیٹر بھی بنایا گیا جس کو اسمبلر (Assembler) کہتے ہیں۔ اس میں لکھا گیا کوڈ انسان کو آسانی سے سمجھ آ سکتا تھا لیکن اس کی رفتار بہت سست تھی اس لئے کہ یہ پہلے مشین لینگویج میں کنورٹ کیا جاتا اور پھر یہ مطلوبہ کام کرتا تھا۔

اس کے بعد پھر High لیول لینگویجز بنائی گئیں۔ ان میں ایک کام آپ صرف ایک سٹیٹ مینٹ میں پرفارم کر سکتے ہیں۔ اس لئے ان کی دوسری لینگویجز کی نسبت سپیڈ بہت زیادہ ہے اور ٹرانسلیٹر پروگرام جو اس لینگویج کو مشین لینگویج میں کنورٹ کرتا ہے۔ وہ کمپائلر کہلاتا ہے۔ ++C لینگویج بھی ایک ہائی لیول لینگویج ہے۔ جو کہ تقریباً سب سے زیادہ پاورفل اور استعمال ہونے والی لینگویج ہے۔ کمپائلر کے بعد پروگرامرز نے انٹرپریٹر پروگرام بنایا جو کہ ہائی لیول لینگویج کوڈ کو ڈائریکٹ ایگزیکیوٹ کرتا ہے۔ اس طرح اسے پہلے کمپائلر کی مدد سے مشین لینگویج میں کنورٹ کرنے کی ضرورت ختم ہوگئی اور اس کی سپیڈ مزید تیز ہوگئی۔

++C کی تاریخ:

1967ء میں مارٹن رچرڈ نے دو لینگویجز BCPL اور B.BCPL لکھیں جو کہ آپریٹنگ سسٹم اور کمپائلر بنانے کے لئے آئیڈیل تصور کی جاتی تھیں۔ 1970ء میں کین تھامپسن نے اپنی لینگویج B میں کئی اہم فیچرز کا اضافہ کیا کیونکہ وہ ایک آپریٹنگ سسٹم بنانا چاہتا تھا۔ اسی سال کے اختتام پر اس نے بیل لیبارٹری میں یونیکس (UNIX) آپریٹنگ سسٹم کا پہلا ورژن تیار کیا۔

پھر اس کے بعد ڈینس رچی (Dennis Ritchi) نے بیل لیبارٹری میں B لینگویج کی مدد سے ایک نئی لینگویج متعارف کروائی جس کا نام C رکھا گیا۔ اور یہ پہلی دفعہ 1972ء میں مارکیٹ میں متعارف کروائی گئی۔ C میں B اور BCPL لینگویجز کے کئی اہم فیچرز شامل ہیں۔ C سے یونیکس آپریٹنگ سسٹم بنایا گیا جس کی وجہ سے یہ بہت مشہور ہوگئی آجکل زیادہ تر آپریٹنگ سسٹم C میں ہی لکھے جا رہے ہیں۔

آپ نے سنا ہوگا کہ انسان نے چاند کو تسخیر کر لیا ہے لیکن پھر بھی اسے سکون نہیں ہے وہ زیادہ سے زیادہ کامیابیاں حاصل کرنا چاہتا ہے۔ اسی طرح پروگرامرز نے بھی محسوس کیا کہ C میں وہ تمام کام نہیں کر سکتے جن کی انہیں ضرورت ہے۔ اس کے پیش نظر 1980ء میں بیل کی لیبارٹری میں بجارن

سٹروسٹرپ (Bjarne Stourstrup) نے C++ لینگوئج بنائی۔ یہ اصل میں C لینگوئج کی ایڈوانس فارم ہے۔ C++ میں C کی نسبت کئی اہم فیچرز شامل کئے گئے اور اس کے مشہور ہونے کی سب سے بڑی وجہ اوبجیکٹ اوورینٹڈ پروگرامنگ ہے۔ اوبجیکٹس ایسے سافٹ ویئر کمپونینٹ ہوتے ہیں جنہیں آپ بار بار استعمال کر سکتے ہیں۔ اوبجیکٹ اوورینٹڈ پروگرام سمجھنے اور دوبارہ ایڈٹ کرنے میں آسان ہوتے ہیں اور یہ بالکل درست کام کرتے ہیں۔

C++ سٹینڈرڈ لائبریری:

آپ نے اوپر C++ کی تاریخ کے بارے میں پڑھا کہ یہ کب اور کس مقصد کے لئے بنائی گئی۔ ہم نے آپ کو بتایا کہ C++ اصل میں C کی ایڈوانس فارم ہے۔ یوں اس میں C کے فیچرز کے علاوہ اضافی فیچرز بھی شامل کئے ہیں۔ C++ پروگرام کلاسسز اور فنکشنز پر مشتمل ہوتے ہیں۔ C++ کی سٹینڈرڈ لائبریری پہلے سے بنے ہوئے فنکشنز اور کلاسسز کی ایک بڑی مقدار فراہم کرتی ہے۔ اس طرح آپ کو C++ میں یہ سیکھنا بہت ضروری ہے کہ اس کی سٹینڈرڈ لائبیریری کس طرح استعمال کر سکتے ہیں۔

C++ پروگرامز کا تعارف:

C++ کے پروگرامز مختلف چھ مراحل میں سے گزر کر آپ کو آؤٹ پٹ ڈسپلے کرتے ہیں۔ یہ مختلف چھ مراحل مندرجہ ذیل ہیں۔

ایڈٹ، پری پراسس، کمپائل، لنک، لوڈ اور ایگزیکیوٹ

پہلا مرحلہ ایک فائل کو ایڈٹ کرنے کا ہے۔ یہ کام آپ ایڈیٹر پروگرام میں کرتے ہیں۔ یعنی اپنا پروگرام ٹائپ کرتے ہیں اور ضروری غلطیاں ختم کرتے ہیں۔ اس کے بعد پروگرام مزید بعد میں استعمال کرنے کے لئے ہارڈ ڈسک پر کہیں بھی سٹور کر دیا جاتا ہے۔ آپ کے C++ پروگرام کی ایکس ٹینشن cpp. ہونا ضروری ہے۔

اس کے بعد پروگرام کمپائل کیا جاتا ہے۔ اس میں C++ کمپائلر C++ کوڈ مشین لینگوئج کوڈ میں ٹرانسلیٹ کرتا ہے۔ اس کے بعد پروگرام کو سٹینڈرڈ لائبریری یا اگر آپ نے کہیں اور سے کلاس یا فنکشن استعمال کیا ہے تو اس کے ساتھ لنک کرنے کا مرحلہ آتا ہے۔

اس کے بعد لوڈنگ پوائنٹ ہے۔ کوئی بھی پروگرام ایگزیکیوٹ کئے جانے سے پہلے میموری میں لوڈ کیا جاتا ہے اور یہ کام لوڈر کرتا ہے اور آخر میں آپ کا پروگرام ایگزیکیوٹ ہوتا ہے۔ اگر اس میں کوئی غلطی نہیں ہوگی تو یہ ایگزیکیوٹ ہوگا ورنہ لنکر یا کمپائلر جب ایرر دے گا تو وہ غلطی درست کریں اور بعد میں اپنا پروگرام ایگزیکیوٹ کریں۔

C++ ایک مشکل لینگوئج ہے اس لئے اس میں آپ کو زیادہ سے زیادہ مہارت حاصل کرنا ہوگی تب آپ ایک اچھا پروگرام لکھ سکیں گے۔ اس کتاب میں ہم نے کوشش کی ہے کہ آپ C++ کی بنیادی چیزیں زیادہ سے زیادہ سیکھ سکیں۔

C++ پروگرام لکھنا:

C++ لینگوئج کمپیوٹر پروگرام کا ایک واضح اور منظم ڈیزائن مہیا کرتی ہے یعنی اس میں لکھے ہوئے پروگرام کا خاکہ بہت منظم ہوتا ہے۔ جیسا کہ ہم نے پہلے بتایا ہے کہ C++ کا کمپائلر سورس فائل کو پراسس (کمپائل) کرنے کے بعد ایگزیکیوٹیبل فائل میں تبدیل کر دیتا ہے جسے آپ اپنے کمپیوٹر میں دوسرے پروگرامز کی طرح چلا سکتے ہیں۔ اس میں شامل سورس فائلز اصل میں ٹیکسٹ فائلز ہوتی ہیں۔ ان کی ایکس ٹینشن cpp. ہوتی ہے جبکہ ایگزیکیوٹیبل (Executable) فائلز کی ایکس ٹینشن exe. ہوتی ہے۔

ہم نے نیچے اس باب میں C++ پروگرامنگ کی کچھ بنیادی مثالیں تحریر کی ہیں جن میں C++ کے اہم بنیادی فیچرز کا استعمال کیا گیا ہے۔ ایک پروگرام کچھ ہدایات (انسٹرکشنز) کا مجموعہ ہوتا ہے۔ جسے ایگزیکیوٹ کیا جا سکتا ہے۔ آیئے C++ کا ایک سادہ سا پروگرام دیکھتے ہیں۔

مثال نمبر Welcome to C++ Programe 1.1

```
# include <iostream.h>
void main( )
    {
    cout <<"Welcome to C++ Program:";
    }
```

یہ ++C کا ایک بنیادی پروگرام ہے۔اس پروگرام کی پہلی لائن یہ ہے۔

```
# include <iostream.h>
```

کسی بھی قسم کی آؤٹ پٹ لکھوانے کے لئے اس لائن کا لکھنا ضروری ہے۔اس میں include # کو پری پروسیسر ڈائریکٹو (Preprocessor Directive) کہتے ہیں۔اس میں # سب سے پہلے پڑھا جاتا ہے۔ یہ ڈائریکٹو ایکسٹرنل فائل iostream.h کو ریفر کرتا ہے۔جس میں cout یعنی ان پٹ/آؤٹ پٹ کے بارے میں معلومات محفوظ ہوتی ہے۔(h.) اس بات کی نشاندہی کر رہا ہے کہ یہ ایک ہیڈر فائل ہے جس میں پہلے سے فنکشنز یا ورڈز کے بارے میں معلومات محفوظ ہیں۔اور ایک اہم بات کہ < اور > بریکٹس اس فائل کے نام کا حصہ نہیں ہیں بلکہ اس کے بعد دوسری لائن void main( ) ہے۔اس کو ہیڈر فنکشن main( ) کہتے ہیں اور یہ ہر ++C پروگرام کے لئے لکھنا ضروری ہے۔ یہ ++C کمپائلر کو یہ واضح کرتی ہے کہ پروگرام کہاں سے شروع ہوتا ہے۔اس میں یہ ( ) بریکٹس لکھنا بھی ضروری ہیں یہ فنکشن کے لئے لکھی جاتی ہیں۔اس کے بعد یہ } بریکٹ ہے جو سب سے آخر میں بند { بھی ہو رہی ہے۔کسی بھی فنکشن کے لئے ان { } بریکٹس کا لکھنا ضروری ہوتا ہے۔ یہ main( ) فنکشن کی باڈی کی نشاندہی کرتی ہیں۔( )main پروگرام کا یہ حصہ ہے اس کے بعد یہ لائن ہے۔

```
cout <<"Welcome to C++ Program:";
```

یہ لائن cout اوبجیکٹ کو یہ "Welcome to C++ Programe" سٹیٹ مینٹ بھیجنے کے لئے استعمال ہو رہی ہے۔آپ اس کی جگہ پر کچھ بھی پرنٹ کروا سکتے ہیں۔اس میں cout سٹینڈرڈ آؤٹ پٹ سٹریم ہے جو کوئی بھی لائن کمپیوٹر سکرین پر ڈسپلے کروانے کے لئے استعمال ہوتی ہے اور cout, (cout output) کا مخفف ہے۔اور آپ دیکھ رہے ہوں گے کہ ہم نے اس میں مطلوبہ الفاظ " " میں تحریر کئے ہیں۔اس سے واضح ہوا کہ اپ جو بھی الفاظ ڈسپلے کروانا چاہتے ہیں انہیں " " میں تحریر کریں وہ بالکل اسی فارمیٹ میں سکرین پر ڈسپلے ہو جائیں گے جس طرح آپ نے لکھے ہیں۔ اور اس کے آخر میں سیمی کالن (;) سے یہ ہر سٹیٹ مینٹ کے بعد لکھنا ضروری ہے اور جب تک آپ یہ نہیں لکھیں گے کمپائلر ہر لائن کو ایک ہی سٹیٹ مینٹ تصور کرے گا یعنی یہ ایک سٹیٹ مینٹ کے اختتام کی نشاندہی کرتا ہے۔cout کے بعد >> علامت کو آؤٹ پٹ آپریٹر یا انسرشن (Insertion) آپریٹر کہتے ہیں۔ یہ آپ کی تحریر cout کو منتقل کرتا ہے۔آپ نے ++C میں پروگرام کے کچھ اہم فیچر کے بارے میں سیکھا۔ یہ تقریباً ہر پروگرام میں استعمال ہوتے ہیں۔آیئے اب اس کی آؤٹ پٹ دیکھتے ہیں۔اس کو کمپائل کریں گے تو یہ آؤٹ پٹ ہوگی۔

```
Welcome to C++ Program:
```

آپ اسی پروگرام کو کئی طریقوں سے لکھ سکتے ہیں مثلاً

```
# include <iostream.h>
void main( )
   {
   cout <<
        "Welcome to C++ Program:";
```

```
}
```

یا

```
void main( )
{
cout
<<
"Welcome" <<"to" <<"C++ Programe";
}
```

یہ مختلف طریقے بتانے کا مقصد صرف یہ ہے کہ آپ کو ++C کے بنیادی فیچرز کے بارے میں معلومات حاصل ہونی چاہئے۔ ان سب پروگرامز کی آؤٹ پٹ ایک ہی ہوگی لیکن صرف تیکنیک مختلف ہے۔

کومینٹس شامل کرنا:

کسی بھی پروگرام میں کومینٹس شامل کرنا ایک اچھے پروگرامر کی صفت جانی جاتی ہے۔ آپ پروگرام میں یہ کومنٹس صرف اپنی سہولت کے لئے شامل کرتے ہیں۔ یہ کومنٹس آؤٹ پٹ کا ایک حصہ نہیں ہوتے ہیں بلکہ یہ آپ کو پروگرام کے متعلق اضافی معلومات فراہم کرتے ہیں اور جب آپ اس پروگرام کو اپڈیٹ (تبدیل) کرنا چاہتے ہیں تو آپ کے لئے مددگار ثابت ہوتے ہیں یعنی آپ کو اس بات کی وضاحت کرتے ہیں کہ پروگرام کی فلاں لائن کا کیا مقصد ہے۔ آپ اپنے پروگرام میں دو طریقوں سے کومنٹس شامل کر سکتے ہیں۔

```
// output statement
```

اور

```
/* output
of Marks
average /*
```

اب یہ پروگرام میں کسی جگہ اور کیسے لکھے جا سکتے ہیں آیئے دیکھتے ہیں۔

```
# include <iostream.h> //header file
void main( ) //main Program
{
cout <<"How Are you Choudhry:";
    /*our Printing Message*/
} /*End of
    Program*/
```

آپ اس طرح اپنے پروگرام میں کومنٹس شامل کر سکتے ہیں۔ آپ سوچ رہے ہوں گے کہ // اور /* */ میں کیا کرتی ہے؟ آپ جب بھی سنگل لائن یعنی صرف ایک لائن کو کومنٹ ٹیکسٹ شامل کرنا چاہتے ہیں تو یہ // استعمال کرتے ہیں جبکہ اگر آپ کو کومنٹ ٹیکسٹ ایک لائن سے زیادہ ہے تو ہر دفعہ // علامت لکھنے کی بجائے آپ ٹیکسٹ کے آغاز پر */ اور اختتام پر /* یہ علامات تحریر کر دیں۔ جو بھی لائن یعنی تحریر ان کے درمیان ہوگی کمپائلر اس کو Read نہیں کرتا۔

ویری ایبلز:

ویری ایبل ایک ایسی علامت ہے جو کمپیوٹر کی میموری میں ڈیٹا محفوظ کرنے کے لئے استعمال ہوتی ہے یا ویری ایبل میموری کے ایک ٹکڑے کا نام ہے جو آپ C++ پروگرام میں انفارمیشن محفوظ کرنے کے لئے استعمال کرتے ہیں۔ میموری کا ہر ٹکڑا جو آپ اپنے پروگرام میں ڈیفائن کرتے ہیں ایک مخصوص قسم کا ڈیٹا محفوظ کرسکتا ہے یعنی جس ٹائپ کا ویری ایبل ہوگا وہ صرف اس ٹائپ کا ڈیٹا سٹور کروانے کے لئے استعمال ہوگا۔

نوٹ: یہ ٹائپ کیا ہے؟ آپ آگے اس باب میں دیکھیں گے۔

مثلاً آپ ایک ویری ایبل نمیریک ویلیو محفوظ کرنے کے لئے لکھتے ہیں تو پھر آپ اس میں کریکٹر یا اعشاری نظام میموری میں محفوظ نہیں کروا سکتے۔ آپ صرف نمبرز 3 ,2 ,1 ہی سٹور کروا سکتے ہیں اور آپ جو معلومات میموری میں محفوظ کریں گے وہ ویری ایبل کی ویلیو ہوگی۔ کسی بھی ویری ایبل کو ویلیو اسی طرح آسائن کی جاتی ہے۔

```
variable = expression/value;
```

کی ورڈز اور ایڈینٹیفائرز: (Keywords & Identifiers):

کی ورڈز کو پروگرامنگ زبان میں ریزرو ورڈز بھی کہتے ہیں یہ ایسے الفاظ ہوتے ہیں جو لینگوئج نے اپنے لئے ریزرو کئے ہوتے ہیں اور یہ خاص مقاصد کے لئے استعمال ہوتے ہیں اور پروگرامر خود سے انہیں دوبارہ ڈیفائن نہیں کرسکتا اور نہ ہی ان کو بطور ویری ایبل استعمال کرسکتا ہے۔ مثلاً cin , cout , count , int وغیرہ

ایسا نام جو آپ ویری ایبل کے لئے استعمال کرتے ہیں یا ایسا نام جو آپ C++ میں کسی بھی اوبجیکٹ کے لئے منتخب کرتے ہیں وہ ایڈینٹیفائر کہلاتا ہے۔ ایک ایڈینٹیفائر (حروفِ تہجی) الفانمیریک کریکٹر کا ایک سٹرنگ ہوتا ہے۔ اس کا پہلا حرف ہمیشہ کریکٹر ہوتا ہے۔ تقریباً کل 53 الفابیٹک کریکٹرز ہیں جن میں 52 حروف اور ایک (-) انڈرسکور ہے۔ ان کے علاوہ 10 نمیریک کریکٹرز بھی ہیں۔ آپ کسی بھی ایڈینٹیفائر کا آغاز C++ آپریٹرز (* ,- ,+) وغیرہ سے شروع نہیں کرسکتے۔ ہم نے پہلے بھی بتایا ہے کہ C++ ایک Case sensitive لینگوئج ہے یعنی جس میں حروفِ تہجی کا لحاظ کیا جاتا ہے یعنی sum اور SUM میں فرق ہے۔

نیولائن کریکٹر:

نیولائن کریکٹر کیا ہے؟ اس پروگرام کو دیکھیں اور بعد میں سمجھنے کی کوشش کریں۔

```
# include <iostream.h>
void main( )
     {
     cout <<"Welcome Mr. Shoaib.\n";
     cout <<"How are you Lala Rukh." <<endl;
     }
```

اس پروگرام کو جب آپ کمپائل کریں گے تو اس کی آؤٹ پٹ کچھ یوں ہوگی۔

```
Welcome Mr. Shoaib
How are you Lala Rukh.
```

پہلی لائن میں n\ علامت نیولائن کریکٹر کہلاتی ہے۔ آپ جب بھی کسی آؤٹ پٹ سٹیٹ مینٹ کے بعد یہ کریکٹر شامل کریں گے تو یہ کمپائلر کو یہ

واضح کرے گا کہ اس کے بعد جو بھی تحریر پرنٹ کرنی ہے وہ اگلی یعنی نئی لائن پر ڈسپلے ہو۔

اسی طرح نیچے والی لائن میں endl لکھا ہوا ہے۔ یہ پہلے سے ڈیفائن کیا ہوا کی ورڈ ہے اور یہ نیو لائن کریکٹر '/n' کا متبادل ہے یہ endline کریکٹر کہلاتا ہے اور یہ بھی نئی لائن کے لئے استعمال ہوتا ہے۔

ڈیٹا ٹائپس:

ہم نے اوپر بیان کیا ہے کہ آپ ایک ویری ایبل میں صرف اسی قسم کا ڈیٹا سٹور کر سکتے ہیں۔ یعنی اس کی ویلیو اسی ٹائپ کی ہوگی جس ٹائپ کا وہ ویری ایبل ہوگا۔ اب یہ ٹائپ کیا ہے؟ ٹائپ میں یہ بات واضح ہوتی ہے کہ آپ کا ویری ایبل کس قسم کا ہے۔ مثلاً

```
int cubes;
```

اس میں cubes ویری ایبل کا نام ہے جبکہ سیمی کالن (;) سٹیٹ مینٹ کا اختتام ہے اور int ویری ایبل کی ٹائپ ہے۔ نیچے ہم نے ایک ٹیبل بنایا ہے جس میں C++ کی بنیادی ٹائپس کے متعلق معلومات ہے۔

C++ ویری ایبل ٹائپس

Table ٹیبل 1.1

Keyword	Low	High	Bytes of Memory
char	–127	127	1
short	–32,767	32,767	2
int	–214,748,3687	214,748,3687	4 or 2
long	–2,147,483,687	2,147,483,687	4
float	3.4×10^{-38}	3.4×10^{38}	4
double	1.7×10^{-308}	1.7×10^{308}	8
long double	1.7×10^{-4932}	1.7×10^{4932}	10

نوٹ: int یونیکس پر 4 بائٹس ریزرو کرتا ہے۔ ویسے اس کا سائز 2 بائٹس بھی ہوتا ہے۔

ان ڈیٹا ٹائپس کو استعمال کرتے ہوئے آپ ان +ve اور –ve دونوں نوعیت کی ویلیوز کو محفوظ کر سکتے ہیں۔ لیکن اگر آپ صرف +ve ویلیوا پنے ویری ایبلز کو آسائن کرنا چاہتے ہیں تو اس کے لئے ان آسائنڈ (Unsigned) ڈیٹا ٹائپس استعمال کی جاتی ہیں اور ان کی رینج بھی تبدیل ہو جاتی ہے۔ نیچے ان ٹائپس کا بھی ایک ٹیبل درج ہے۔

C++ ان سائنڈ (Unsigned) ڈیٹا ٹائپس

Table ٹیبل 1.2

Keyword	Low	High	Bytes of Memory
unsigned char	0	255	1
unsigned short	0	65,535	2

unsigned int	0	4,294,967,295	4
unsigned long	0	4,294,976,295	4

ان ڈیٹا ٹائپس کی مدد سے آپ ویری ایبلز کو مطلوبہ ویلیو آسائن کر سکتے ہیں۔ان ڈیٹا ٹائپس کو آپ کس طرح استعمال کر سکتے ہیں۔آپ آگے اس باب میں تفصیل سے پڑھیں گے۔

integer ڈیٹا ٹائپ:

ایک integer ایک مکمل نمبر ہوتا ہے مثلاً 3-, 2-, 1-, 3, 2, 1 وغیرہ۔اب آئیے دیکھتے ہیں کہ اس قسم کی ٹائپ میں کون کون سی ڈیٹا ٹائپس آپ استعمال کر سکتے ہیں۔

byte , short , int , long

نوٹ: Borland C++ میں int کی ویلیو 32768- سے 32768 یعنی 2 بائٹس ہوتی ہے۔

آپ جب بھی integer ویلیو میموری میں سٹور کرنا چاہتے ہیں تو ان میں سے کوئی بھی ڈیٹا ٹائپ استعمال کر سکتے ہیں۔اس کے علاوہ اس کی unsigned ٹائپس بھی ہیں جن کا ذکر ہم نے اوپر کیا ہے۔ان ٹائپس کے ویری ایبلز کو آپ یوں ڈیکلئر کر سکتے ہیں۔

```
byte        small;
short       medium;
int         number;
long        bigral;
```

یہ اوپر ویری ایبلز ڈیکلئر کرنے کا طریقہ ہے یعنی کہ number کی ڈیٹا ٹائپ int ہے۔اب number ویری ایبل int ٹائپ کا ہے اور آپ اس کو بھی int ٹائپ کا ڈیٹا آسائن کر سکتے ہیں۔اب جب بھی یہ سٹیٹ مینٹ کمپائل ہوگی تو میموری میں اس کے لئے دو بائٹس ریزرو کر دی جائیں گی۔اب اس کے بعد مرحلہ ویری ایبل انی شلائیزیشن کا ہے۔آپ کسی بھی ویری ایبل کو یوں انی شلائیز کر سکتے ہیں۔

```
int number = 81;
```

یا اس کے علاوہ ایک طریقہ یہ بھی ہے۔

```
int number, a;
number = 10'
a = 101;
```

اسی طرح آپ کسی بھی ٹائپ کا ویری ایبل انی شلائیز کر سکتے ہیں۔

```
long    bigral;
bigral = a+number;
bigral = 10131
```

آئیے ایک پروگرام دیکھتے ہیں جس میں ویری ایبلز کا استعمال کیا ہوا ہے۔

مثال نمبر 1.2

```
# include <iostream.h>
    //This program shows initialisation of variables.
```

```
void main( )
  {
  int a;        //Declaration
  short b;
  long sum;
  a = 10;       //initialisation
  b = 7;
  Sum = a+b;
  cout <<"The Sum of:" <<a <<"and" <<b
       <<"is" <<"Sum";
  }
```

اب اس پروگرام میں یہ ہو رہا ہے کہ سب سے پہلے تین ویری ایبلز a جس کی ٹائپ int ہے اور b کی ٹائپ short اور long ٹائپ کا Sum ویری ایبل ڈیکلیر کیا ہے بعد میں a اور b کو ویلیو آسائن کی ہے ہے یعنی انی شلائیز کیا ہے اور بعد میں Sum میں a + b کروایا ہے یعنی ان دونوں کا رزلٹ Sum میں سٹور ہو گا۔ جب آپ اس پروگرام کو کمپائل کریں گے تو اس کی آؤٹ پٹ یہ ہو گی۔

```
The Sum of: 10 and 7 is: 17
```

کریکٹر ڈیٹا ٹائپ:

C++ میں کریکٹرز کو سٹور کرنے کے لئے char ڈیٹا ٹائپ استعمال کی جاتی ہے۔ آپ اس میں کریکٹر کی ہر ویلیو کے گرد 'A' (سنگل کوما کوٹیشن) لگاتے ہیں۔ یہ کریکٹر کو سٹرنگ سے مختلف ظاہر کروانے کے لئے لگایا جاتا ہے۔ یہ میموری میں 8 bits یعنی 1 بائیٹ جگہ گھیرتا ہے۔ یہ integer ٹائپ کا ایک حصہ ہے اس لئے آپ کریکٹر میں نمریک ویلیو بھی تحریر کر سکتے ہیں۔ مثلاً

```
char d = 10;
char b = 'A';
```

char سے مراد کریکٹر ہے آپ جب بھی char ڈیٹا ٹائپ لکھتے ہیں تو اس ٹائپ کا ویری ایبل بطور کریکٹر انٹر پریٹ کیا جاتا ہے۔ اور سسٹم خود بخود اس کا ASCII کوڈ میموری میں سٹور کرتا ہے۔ آیئے اس کی ایک مثال دیکھتے ہیں۔

مثال نمبر 1.3

```
# include <iostream.h>
# include <conio.h>
void main( )
  {
  char variable = 'Z';
  char a = 41;
  cout <<"output is:" <<a <<endl
  <<"and" <<variable;
```

```
getch( );
}
```

اس طرح سے آپ اپنے پروگرام میں کریکٹر استعمال کر سکتے ہیں۔ آپ حیران ہوں گے کہ اس پروگرام میں ایک لائن کا اضافہ ہو گیا ہے getch(); یہ کیا ہے؟ یہ C++ کا پہلے سے بنا ہوا ایک فنکشن ہے۔ اس کا فائدہ یہ ہے کہ جب آپ پروگرام کی آؤٹ پٹ دیکھنے کے لئے اسے کمپائل کریں گے تو وہ فوراً واپس C++ سورس کوڈ پر آ جاتا ہے لیکن اس کے لکھنے سے وہ اس وقت تک سورس ونڈوز میں واپس نہیں آئے گا جب تک آپ کی بورڈ سے کوئی کی (Key) پریس نہیں کریں گے۔ getch ( ); کا مطلب ہے get character یعنی ایک کریکٹر پریس کریں لیکن اس سے متعلق معلومات iostream.h میں سٹور نہیں ہیں۔ اس کے لئے آپ کو (conio.h) لائبریری استعمال کرنی پڑتی ہے۔

فلوٹنگ پوائنٹ ڈیٹا ٹائپس:

ایسی ویلیوز جن میں آپ نے اعشاری نظام استعمال کیا ہو اور وہ integer ٹائپس میں استعمال نہیں کی جا سکتی ہوں وہ فلوٹنگ پوائنٹ ڈیٹا ٹائپس میں محفوظ کی جاتی ہیں۔ مثلاً 3.14 , 11.561 وغیرہ۔

اس کے لئے آپ float اور double استعمال کرتے ہیں۔

مثال نمبر 1.4

```
float a;
float a = 3.14;
```

اس کو ایک مثال سے دیکھتے ہیں۔

```
//floating point variables
void main( )
    {
    double result;
    float a = 3.14;
    double d = 5.69;
    result = a×d;
    cout <<"Answer is:" <<result;
    getch( );
    }
```

حسابی (Arithmetic) آپریٹرز:

ایک آپریٹر ایسی علامت ہوتی ہے جو ایک یا ایک سے زائد ایکسپریشن کو آپریٹ کر کے ایک ویری ایبل کو اس کی ویلیو اسائن کرتی ہے۔ آپ نے اس کتاب میں آؤٹ پٹ آپریٹر >> کے بارے میں پہلے ہی پڑھا ہے اس کے علاوہ ایک اہم آپریٹر اسائنمنٹ (=) ہے جو ایک ویلیو ویری ایبل کو اسائن کرنے کے لئے استعمال ہوتا ہے۔ یہ بائیں طرف والی ویلیو دائیں طرف کے ویری ایبل کو اسائن کرتا ہے۔ اس کے علاوہ چھ اہم integer حسابی آپریٹرز یہ ہیں۔

Table 1.3 ٹیبل

مثال	وضاحت	آپریٹرز
a+b	جمع	+
a–b	تفریق	–
a*b	ضرب	*
a/b	تقسیم	/
a%b	بقایا	%
–a	منفی	–

ان آپریٹرز کو C++ میں کیسے استعمال کیا جا سکتا ہے؟ اس کی ایک مثال نیچے درج ہے۔

<u>مثال نمبر 1.5 آپریٹرز کا استعمال</u>

```
//use of Arithmetic operators
void main( )
    {
    int a=5, b=3;
    cout <<a <<"+" <<b <<"=" <<(a+b)<<endl;
    cout <<a <<"-" <<b <<"=" <<(a-b)<<endl;
    cout <<a <<"*" <<b <<"=" <<(a*b)<<endl;
    cout <<a <<"/" <<b <<"=" <<(a/b)<<endl;
    cout <<a <<"%" <<b <<"=" <<(a%b)<<endl;
    getch( );
    }
```

اس پروگرام میں ہم نے پیچھے والے پروگرام کی نسبت قدرے مختلف طریقے سے رزلٹ پرنٹ کروایا ہے تا کہ آپ کو C++ پروگرامنگ کے تمام فیچرز کے بارے میں معلومات حاصل ہو جائے۔ آپ اس کے علاوہ a اور b کا رزلٹ کسی ویری ایبل میں بھی محفوظ کر سکتے ہیں۔ مثلاً

```
c = a+b
c = a%b
```

یا

اس کے علاوہ آپ نے دیکھا کہ اوپر ہم نے ہیڈر فائل iostream.h نہیں لکھیں وہ آپ خود لکھیں گے۔ اس پروگرام کی آؤٹ پٹ یہ ہوگی۔

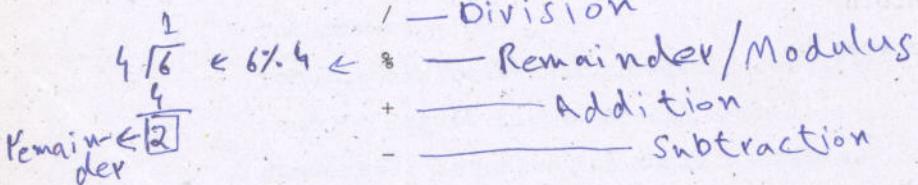
```
a+b = 8
a-b = 2
a×b = 15
a/b = 1
```

```
a%b = 2
```

اس میں a/b پر غور کریں اس کا رزلٹ 1 ہے اس کی وجہ یہ ہے کہ اس کا رزلٹ اعشاری نظام میں آتا ہے لیکن ہم نے ان کو int ڈیکلیر کیا ہے۔ اس لئے وہ اعشاری نظام کو نظر انداز کر دے گا۔ اس کے لئے آپ کو ٹائپ کنورژن کرنا ہوگی جو بعد میں آپ پڑھیں گے۔

آپریٹرز کی فوقیت: (Operator Precedence):

ایک ایکسپریشن میں ایک سے زیادہ آپریٹرز بھی موجود ہو سکتے ہیں۔ لہٰذا ہمیں اس بات کے بارے میں معلومات ہونی چاہئے کہ ایکسپریشن میں کون سا آپریٹر سب سے پہلے پرفارم ہوگا۔ آپریٹرز اس طرح Evaluate ہوتے ہیں یعنی ایکسپریشن میں آپریٹرز اس ترتیب سے حل کئے جاتے ہیں۔

```
Highest to lowest
( )     [ ]
*
/
%
+
-
<<
=
```

یونری آپریٹرز:

C++ C لینگوئج کی طرح ++ اور -- آپریٹر بھی استعمال کرتی ہے۔ ان میں سے ++ کو اضافی اور -- کمی کے آپریٹر کہتے ہیں۔ یعنی increment اور decrement آپریٹرز ان میں ++ اپنے آپرینڈ میں ایک اضافہ کرتا ہے جبکہ -- اپنے آپرینڈ میں ایک کمی کرتا ہے کیونکہ ان آپریٹرز کا آپرینڈ صرف ایک ہوتا ہے اس لئے انہیں یونری آپریٹرز بھی کہتے ہیں۔ مثلاً

```
a++;   یا   ++a;
a--;   یا   --a;
```

یعنی آپ یہ آپریٹرز ویری ایبل کے دونوں طرف لگا سکتے ہیں لیکن دونوں صورتوں میں رزلٹ مختلف ہوگا۔ اس کو پوسٹ فکس اور پری فکس کہتے ہیں۔ پوسٹ فکس میں آپرینڈ کی ویلیو حاصل کرنے کے بعد اس میں ایک کا اضافہ ہوتا ہے جبکہ پری فکس میں پہلے ایک کا اضافہ ہوتا ہے اور بعد میں آپرینڈ کی ویلیو حاصل کی جاتی ہے۔ اس کو آپ مثال سے بآسانی سمجھ لیں گے۔ پہلے ہم نے ایک سادہ کوڈ لکھا ہے۔

مثال نمبر 1.6

```
{
int a=7, b=8;
    ++a;
    --b;
cout <<"a=" <<a <<",b="<b <<endl;
    a++;
```

```
    b--;
  cout <<"a=" <<a <<",b=" <<b;
  getch( );
  }
```

جب آپ اس پروگرام کو ایگزیکیوٹ کریں گے تو یہ آؤٹ پٹ ڈسپلے ہوگی۔

```
a=8, b=7
a=9, b=6
```

آیئے اب ہم ایک اور پروگرام لکھتے ہیں جو اس کی مکمل وضاحت کرے گا۔

مثال نمبر 1.7

```
# include <conio.h>
# include <iostream.h>
void main( )
  {
  int a=11, b;
  b=++a;
  cout <<"a=" <<a <<",b=" <<b <<endl;
  b=--a;
  cout <<"a=" <<a <<",b=" <<b <<endl;
  b=a--;
  cout <<"a=" <<a <<",b=" <<b <<endl;
  getch( );
  }
```

اب اس پروگرام کو آپ ٹائپ کریں اور ایگزیکیوٹ کریں۔ اس کا رزلٹ یہ ہوگا۔

```
a=12 , b=12
a=11 , b=11
a=11 , b=10
```

آرتھمیٹک آسائنمنٹ آپریٹرز:

C++ میں آپ ایک کوڈ کو کئی طریقوں سے لکھ سکتے ہیں۔ اسی طرح ان میں سے ایک ارتھمیٹک آسائنمنٹ آپریٹر ہے جس کی مدد سے آپ آپریٹرز شارٹ کٹ طریقے سے استعمال کر سکتے ہیں۔ اس سے کوڈ آسان ہو جاتا ہے۔ کوڈ کم ٹائپ کرنے کی ضرورت ہوتی ہے جس سے ٹائم بچ جاتا ہے۔ مثلاً

```
a = a+4;
```

اس سٹیٹ مینٹ کو آپ یوں بھی لکھ سکتے ہیں۔

```
a+ = 4;
```

اس سٹیٹ مینٹ میں آپ a ویری ایبل میں 4 جمع کرنے کے بعد نئی ویلیو a میں سٹور کر رہے ہیں اور یہی کام اوپر والی سٹیٹ مینٹ میں ہو رہا ہے۔ آیئے اس کی ایک مثال دیکھتے ہیں۔

مثال نمبر 1.8 ارتھمیٹک آسائنمنٹ آپریٹر

```
# include <iostream.h>
# include <conio.h>
void main( )
    {
    int result = 5;
    int answer = 7;
    result+=5;//eqivalent to result=result+5;
    cout <<"Result is:" <<result <<endl;
    answer%=2;//equalent to answer=answer%2;
    cout <<"Answer is:" <<answer;
    getch( );
    }
```

اب اس پروگرام کو ایگزیکیوٹ کریں۔ اس کی آؤٹ پٹ بالکل ویسے ہی ہوگی جس طرح آرتھمیٹک آپریٹرز سے آپ پروگرام حل کرتے ہیں۔

آؤٹ پٹ

```
Result is : 10
Answer is : 1
```

ریلیشنل آپریٹرز:

یہ آپریٹرز دو ویلیوز کا موازنہ کرنے کے لئے استعمال ہوتے ہیں۔ یہ ویلیوز C++ کی ڈیٹا ٹائپس میں سے ہونی چاہئے۔ اس کے علاوہ یہ آپریٹرز دونوں آپرینڈ کے درمیان تعلق کو ظاہر کرتے ہیں یعنی یہ دونوں آپرینڈ کی برابری پر ان کا آرڈر واضح کرتے ہیں۔ ان آپریٹرز کا ٹیبل نیچے درج ہے۔

آپریٹرز	وضاحت		
==	برابر	Equal to	a==b;
!=	برابر نہیں ہے	Not equal to	a!=b;
<	چھوٹا ہے	Less than	a<b
>	بڑا ہے	Greater than	a>b
<=	چھوٹا ہے یا برابر ہے	Less than or equal to	a<=b
>=	بڑا ہے یا برابر ہے	Greater than or equal to	a>=b

یہ آپریٹرز ہم لوپس یا if کی کنڈیشن میں استعمال کرتے ہیں۔ جو آپ اگلے باب میں پڑھیں گے۔ یعنی کہ اگر یہ اس کے برابر ہے تو یہ آؤٹ پٹ ہونی چاہئے ورنہ کوئی اور آؤٹ پٹ ہوگی۔ لیکن آیئے دیکھتے ہیں کہ ان کو کیسے استعمال کیا جا سکتا ہے۔

مثال نمبر 1.9 ریلیشنل آپریٹرز

```
{
int num = 5;
cout <<"sum is less 10;:" <<(num<10)<<endl;
cout <<"sum is equal to 10;:" <<(num==10)<<endl;
cout <<"sum is greater 10;:" <<(num>10)<<endl;
getch( );
}
```

اب اس پروگرام کو ذرا سمجھنے کی کوشش کریں۔ اس میں ایک ویری ایبل num ہے جس کی ویلیو 5 ہے۔ اس کے بعد یہ کنڈیشن ہے کہ کیا یہ نمبر 10 کے برابر ہے یا اس سے بڑا ہے اور یا پھر چھوٹا ہے۔ یہ نمبر 10 سے چھوٹا ہے اس لئے جہاں چھوٹا ہے (>) کنڈیشن ہوگی اس کی آؤٹ پٹ 1 ہوگی باقی کا رزلٹ 0 ہوگا۔ اس سے ظاہر ہوا کہ جب بھی ہماری کنڈیشن یا دو نمبرز کا موازنہ درست ہوگا تو رزلٹ 1 شو ہوگا ورنہ صفر ڈسپلے ہوگا۔

ٹائپ کنورژن:

ٹائپ کنورژن کی ضرورت اس وقت پیش آتی ہے جب آپ ایک ڈیٹا ٹائپ کی ویلیو کسی دوسری مختلف قسم کی ڈیٹا ٹائپ کے ویری ایبل کو آسائن کرنا چاہتے ہوں۔ لیکن آپ کو C++ میں اتنا زیادہ فکرمند ہونے کی ضرورت نہیں ہے کیونکہ C++ میں عام طور پر یہ خود بخود عمل ہو جاتا ہے۔ مثلاً

```
int c=7;
float a=3.142;
double res=c×a;
cout <<"Answer is:" <<res;
```

اب اس پروگرام کو اگر آپ ایگزیکیوٹ کریں گے تو اس میں کوئی ایرر نہیں ہوگا کیونکہ کمپائلر کو معلوم ہے کہ آپ c کو a سے ضرب دینا چاہتے ہیں اور وہ خود بخود c کی ویلیو کو 7 تصور کرے گا۔

اس کے علاوہ اگر آپ ٹائپ کاسٹنگ کرنا چاہتے ہیں تو یہ بہت آسان ہے۔ اس لئے پہلے آپ کو یہ طریقہ اختیار کرنا ہوگا۔

```
variable = int(expr);
```

اس سٹیٹ میں expr کی ویلیو int میں کنورٹ ہوگی اور ویری ایبل کو آسائن کر دی جائے گی لیکن فرض کریں کہ آپ کا پہلا نمبر یعنی exp میں اعشاری نمبر ہے یعنی 4.19 ہے تو یہ اس کا اعشاریہ اور بعد والی ویلیو ختم کر دے گا اور صرف 4 ویری ایبل کو آسائن ہوگا۔ آیئے اس کی ایک مثال دیکھتے ہیں۔

مثال نمبر 1.10 ٹائپ کاسٹنگ پروگرام

```
void main( )
{
double d=3415.3124;
int i=int(d);
```

```
cout <<"d" <<d;
cout <<endl <<"i=" <<i;
getch( );
}
```

اب اس پروگرام میں یہ ہورہا ہے کہ ڈبل وری ایبل d کی ویلیو کو int ٹائپ کے i وری ایبل میں سٹور کیا گیا ہے۔ d کی ٹائپ کاسٹنگ بھی کی ہے۔ اس سے d کا اعشاریہ ختم ہوجائے گا یوں اس پروگرام کی آؤٹ پٹ ہوگی۔

```
d = 315.3124
i = 3415
```

ایک بات ذہن نشین کرلیں کہ جب چھوٹی ٹائپ کا وری ایبل بڑی ٹائپ کے وری ایبل میں کنورٹ کیا جاتا ہے تو کسی آپریٹر کی ضرورت نہیں ہوتی۔

ان پٹ:

آپ نے اب سے پہلے وری ایبلز ڈیکلیر کرنا سیکھا اور ویلیوز ان کو خود آسائن کرتے تھے لیکن آپ اس کے علاوہ یہ ان پٹ رن ٹائم پر یوزر سے بھی لے سکتے ہیں اور یہ کوئی مشکل کام بھی نہیں ہے۔ اس کے لئے آپ cin سٹریم استعمال کرسکتے ہیں اور یہ console ان پٹ کا مخفف ہے۔ اب اس کو کب اور کیسے استعمال کرنا ہے آیئے دیکھتے ہیں۔

مثال نمبر 1.11 ان پٹ پروگرام

```
void main( )
{
int number;
char ch;
cout <<"Enter a number:";
cin>> number;
cout <<"Enter a Character:";
cin>> ch;
cout <<"number=" <<number <<"character" <<ch;
getch( );
}
```

اس میں ہم نے cin کے بعد >> آپریٹر استعمال کیا ہے۔ اس کو ان پٹ Extraction آپریٹر بھی کہتے ہیں۔ یہ cin سٹریم کے ساتھ کی بورڈ سے ان پٹ حاصل کرنے کے لئے استعمال کیا جاتا ہے۔ اس میں پروگرام کو جب آپ ایگزیکیوٹ کریں گے تو پہلی لائن پرنٹ ہونے کے بعد سسٹم رک جائے گا اور جیسے ہی آپ کوئی نمبر درج کریں گے وہ وری ایبل کو آسائن کردیا جائے اور پروگرام یا سسٹم دوسری ان پٹ مانگے اور بعد میں اس کا رزلٹ شو کروا دے گا۔ آپ اسکے علاوہ ایک لائن میں ایک سے زائد ان پٹس بھی لے سکتے ہیں۔ مثلاً

```
char ch, ch1;
```

```
cin>> ch>> ch1;
```

سٹرنگ ان پٹ:

اوپر آپ نے cin کی مدد سے ان پٹ لینے کا طریقہ سیکھا ہے لیکن cin کے ساتھ ایک پرابلم ہے کہ اس کی مدد سے آپ سٹرنگ ان پٹ احسن طریقے سے نہیں لے سکتے ہیں۔ اس میں لاجیکل ایرر ہوتا ہے۔ یہ دراصل صرف نمریک ویلیوز کے لئے بہترین ہے یعنی آپ ایک سٹرنگ لکھنا چاہتے ہیں۔ جس میں کافی جگہ (سپیس) ہو تو cin اس میں لاجیکل ایرر پیدا کرتا ہے۔ یہ سپیس کو ایک الگ کریکٹر سٹرنگ تصور کرتا ہے۔ اس کے لئے آپ C++ کا اہم فیچر gets() استعمال کر سکتے ہیں۔ gets اصل میں get سٹرنگ کا مخفف ہے اب اس کو پروگرام میں لکھ سکتے ہیں۔

```
cout <<"Enter you name:";

gets(name);
```

name آپ کا ویری ایبل ہے۔ اس کے لئے آپ کو ایک اور کام کرنا ہوگا کہ ڈائریکٹو شامل کرنا ضروری ہے یعنی اس کی اضافی معلومات iostream میں نہیں ہے اس کے لئے * <stdio.h> ہیڈر فائل استعمال ہوئی ہے۔

```
# include <stdio.h>
```

اس کتاب کے حوالہ سے ایک بات بہت اہم ہے کہ ہم نے اس میں ہر مثال میں ہیڈر فائلز شامل نہیں کیں وہ آپ خود لکھیں گے اور اسکے main() میتھڈ میں clrscr() اور آخر پر getch() ضرور لکھا کریں۔ clrscr() سے آپ کی آؤٹ پٹ سکرین صاف ہو جاتی ہے۔ getch() آؤٹ پٹ سکرین کو اس وقت ختم نہیں کرتا جب تک کہ آپ کی بورڈ سے کوئی کی نہ دبائیں۔ اگر آپ کو کسی کی ورڈ یا پہلے سے بنے ہوئے فنکشن کی ہیڈر فائل کا علم نہیں ہے تو کرسر اس کے نیچے لائیں اور CTRL+F1 دبائیں اور اس سے متعلقہ تمام معلومات کھل جائیں گی۔

سوال نمبر1: مختصر جواب دیں۔

(i) C++ پروگرام ایگزیکیوٹ ہونے سے پہلے کن مراحل میں سے گزرتا ہے؟

(ii) C++ کی اہم ڈیٹا ٹائپس کون کون سی ہیں؟

(iii) cout اور cin میں کیا فرق ہے؟

سوال نمبر2: C++ کا سب سے چھوٹا ایگزیکیوٹیبل پروگرام لکھیں (خواہ وہ کچھ آؤٹ پٹ دے یا نہ دے لیکن کسی ایرر کے بغیر ایگزیکیوٹ ہو)

سوال نمبر3: ایک پروگرام لکھیں جو یوزر سے تین نمبر بطور ان پٹ لے پھر ان کا مجموعہ، اوسط اور تینوں نمبرز کی پراڈکٹ معلوم کریں۔

نوٹ: آپ نے وہی تکنیکس استعمال کرنی ہیں جو اس باب میں پڑھی ہیں۔

سوال نمبر4: اس مساوات کو حل کریں۔ $Z = ax^3 + 3$

فرض کریں کہ a کی ویلیو 4 اور x کی ویلیو 2 ہے۔

سوال نمبر5: فرض کریں کہ ایک کیوبک فٹ میں 3.41 گیلنز آتے ہیں۔ اب آپ یوزر سے گیلنز کی ویلیو مانگیں یعنی یوزر سے ان پٹ لیں اور پھر اس کے مساوی کیوبک فیٹ میں ویلیو شو کروائیں۔

سوال نمبر6: ایک ایسا پروگرام لکھیں جو یہ آؤٹ پٹ شو کروائے۔

```
7
14
15
```

نوٹ: اس میں آپ نے a = 7 کو کانسٹنٹ ڈیکلیر کرنا ہے۔

جوابات

❶: جواب

(i) C++ پروگرام ایگزیکیوٹ ہونے سے پہلے مندرجہ ذیل چھ مراحل میں سے گزرتا ہے۔

(i)	ایڈٹ	(ii)	پری پراسس
(iii)	کمپائل	(iv)	لنک
(v)	لوڈ	(vi)	ایگزیکیوٹ

(ii) C++ کی اہم ڈیٹا ٹائپس مندرجہ ذیل ہیں۔

```
long double, double, float, long, int, short, char
unsigned long, unsigned int, unsigned short, unsigned char
```

(iii) cout<< یہ C++ کا آؤٹ پٹ insertion اپریٹر ہے۔ یہ console output کا مخفف ہے اور جب بھی آپ کسی لائن کی آؤٹ پٹ دیکھنا چاہتے ہیں تو وہ cout میں لکھتے ہیں۔

cin>>: یہ C++ کا ان پٹ آپریٹر ہے یعنی یہ یوزر سے ان پٹ لینے کے لئے استعمال ہوتا ہے اور یہ cousole input کا مخفف ہے۔

❷: جواب

C++ کا سب سے چھوٹا ایگزیکیوٹیبل پروگرام یہ ہے۔

```
void main(void)
  {
  }
```

❸: جواب

```
void main(void)
  {
  clrscr( );
  int a,b,c, sum, pro;
  float avg;
  cout <<In Enter Three integer values:";
  cin>> a>> b>> c;
  sum = a+b+c;
  pro = a*b*c;
  avg = float(sum)/3 //type casting
  cout <<"\n Sum=" <<sum;
```

```
cout <<"\n Product=" <<pro;
cout <<"\n Average=" <<avg;
getch( );
}
```

❹: جواب $Z = ax^3 + 3$

```
void main(void)
{
clrscr( );
cout <<"\n Solving Z=ax³+3:";
int a=4, x=2;
int Z=0;
Z = a*(x*x*x)+3;
cout <<"\n Answer=" <<Z;
getch( );
}
```

❺: جواب

```
void main(void)
{
clrscr( );
int gallons;
float ans;
cout <<"\n Enter value of gallons:";
cin>> gallons;
ans = gallons/3.41;
cout <<"\n gallons <<"in cubic feet:" <<ans;
getch( );
}
```

❻: جواب

```
void main(void)
{
clrscr( );
```

```
const int a=>;
int ans;
cout <<ans <<endl;
ans = a+a;
cout <<ans <<endl:
ans ++;
cout <<ans <<endl;
ans- =2
cout <<ans;
getch( );
}
```

باب نمبر 2

کنٹرول سٹرکچر اینڈ فنکشنز

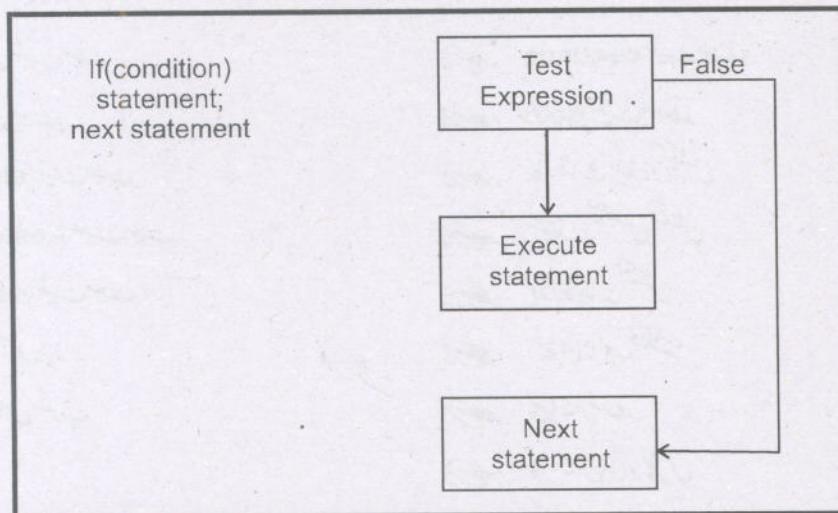
اکثر آپ نے دیکھا ہوگا کہ کام ایک ہی فلو میں شروع سے آخر تک ختم نہیں ہوتے بلکہ آپ اس کا کچھ حصہ پہلے اور کچھ بعد میں مکمل کرتے ہیں۔ اسی طرح زیادہ طرح پروگرامز بھی شروع سے آخر تک ایک ہی آرڈر میں ایگزیکیوٹ نہیں ہوتے۔ بعض اوقات ایگزیکیوشن کنٹرول پروگرام کے ایک حصے سے دوسرے کو ٹرانسفر کر دیا جاتا ہے۔ ایسی سٹیٹ مینٹس جو کنٹرول ایک لائن سے دوسری کو دیتی ہیں وہ کنٹرول سٹرکچر یا کنٹرول سٹیٹ مینٹس کہلاتی ہیں۔ یہ مندرجہ ذیل ہیں۔

لوپس کنٹرول سٹیٹ مینٹ

فنکشن ایک پروگرام کی کئی سٹیٹ مینٹس کو ایک یونٹ میں اکٹھا کرتا ہے اور ان سٹیٹ مینٹس کو ایک مخصوص نام دیتا ہے اور آپ پھر یہ گروپ پروگرام کے کسی بھی حصہ سے صرف گروپ کا نام کال کر سکتے ہیں۔ فنکشن استعمال کرنے کا ایک فائدہ یہ ہے کہ آپ کے پروگرام کا سائز کم ہو جاتا ہے۔ پروگرام کوڈ تھوڑا ہوگا تو آپ کا ٹائم بھی بچے گا اور کوڈ سمجھنا بھی آسان ہوگا۔ فنکشن کوڈ میموری میں صرف ایک جگہ محفوظ ہوتا ہے خواہ یہ فنکشن کئی دفعہ ہی کاپی کیوں نہ کیا گیا ہو۔ اسی باب کے پہلے حصہ میں آپ کنٹرول سٹیٹ مینٹس اور لوپس کے بارے میں پڑھیں گے جبکہ اس باب کے دوسرے حصہ میں آپ کو فنکشن کے بارے میں بتایا گیا ہے۔ یہ باب بڑی اہمیت کا حامل ہے۔ اس میں آپ یہ ٹاپکس پڑھیں گے۔

- کنٹرول سٹیٹ مینٹ
- if سٹیٹ مینٹ
- else-if سٹیٹ مینٹ
- Nested-if سٹیٹ مینٹ
- switch سٹیٹ مینٹ
- لاجیکل آپریٹرز
- ویری ایبل سکوپ
- لوپس
- for لوپ
- Nested-for لوپ
- while لوپ
- do-while لوپ
- بریک سٹیٹ مینٹ
- continue سٹیٹ مینٹ
- goto سٹیٹ مینٹ
- کانسٹنٹ اینڈ آبجیکٹس
- سٹینڈرڈ میتھس فنکشن
- یوزر ڈیفائن فنکشن
- ریٹرن ٹائپ فنکشنز
- پیرامیٹر لسٹ
- پیرامیٹر بائی ریفرنس
- ریکرژن
- فنکشن اوور لوڈنگ
- ڈیفالٹ آرگومنٹس
- مشق

کنٹرول سٹیٹ مینٹ:

کسی بھی قسم کا فیصلہ کرنا یا کسی بھی چیز کا انتخاب کرنا آپ کے پروگرام کا ایک بنیادی حصہ ہوتا ہے۔ مثلاً اس نوعیت کے فیصلے آپ اپنے پروگرام میں کرتے ہیں کہ اگر یوزر کا نام اور پاس ورڈ درست ہے تو وہ پروگرام میں داخل ہو گا ورنہ پہلے درست پاس ورڈ تحریر کرے۔ اگر استعمال کنندہ کے اکاؤنٹ میں پیسے ہیں تو وہ شاپنگ کر سکتا ہے ورنہ نہیں۔ اسی طرح سٹوڈنٹس کی اوسط اور ان کا گریڈ وغیرہ معلوم کرنا۔ پروگرامنگ میں آپ ایسا کرنے کے لئے ویری ایبلز، کانسٹینٹس اور ایکسپریشنز کو چیک کرتے ہیں۔ ++C میں اس کے لئے دو اقسام کی سٹیٹ مینٹس استعمال کی جاتی ہیں۔

```
(i)     The if Statement
(ii)    The else-if statement
```

if سٹیٹ مینٹ:

اس سٹیٹ مینٹ کی مدد سے آپ اپنی ایکسپریشن چیک کرتے ہیں کہ کیا وہ درست ہے یا نہیں؟ اور اگر درست ہے تو کیا رزلٹ شو کروائے گی۔ اس کا جنرل سینٹیکس (syntax) یہ ہے۔

```
if (expression/condition)
        statement;
```

اب یہ آپ کی کنڈیشن کسی بھی نوعیت کی ہو سکتی ہے اگر وہ درست ہو گی تو یہ سٹیٹ مینٹ ایگزیکیوٹ ہو گی ورنہ یہ ایگزیکیوٹ نہیں ہو گی۔ اس کی حرآرکی سے آپ یہ بات سمجھ سکتے ہیں۔

اس میں آپ آسانی سے سمجھ سکتے ہیں کہ اگر ایکسپریشن True (درست) ہو گی تو اس کے فوراً بعد والی سٹیٹ مینٹ ایگزیکیوٹ ہو گی اور اگر ایکسپریشن/کنڈیشن false (غلط) ہو گی تو باہر والی سٹیٹ مینٹ ایگزیکیوٹ ہو گی۔ اس صورت میں ہماری if کنڈیشن میں صرف ایک سٹیٹ مینٹ ہے لیکن if کے بعد ایک سے زائد سٹیٹ مینٹس بھی لکھی جا سکتی ہیں۔ اس کے لئے { } بریکٹس کا استعمال کیا جاتا ہے۔ مثلاً

```
if (condition)
   {
```

```
statement 1;
statement 2;
:
statement n;
}
```

اوپر ہم نے حرآرکی میں بریکٹس استعمال نہیں کی اس لئے یہاں کنڈیشن درست ہونے کی صورت میں صرف پہلی سٹیٹ مینٹ اس کنڈیشن کے تحت ایگزیکیوٹ ہوگی جبکہ اس طریقہ کار میں بریکٹس میں تمام سٹیٹ مینٹس ایگزیکیوٹ ہوں گی۔ آیئے اب اس کی ایک مثال دیکھتے ہیں۔

مثال نمبر 2.1 if سٹیٹ مینٹ کا استعمال

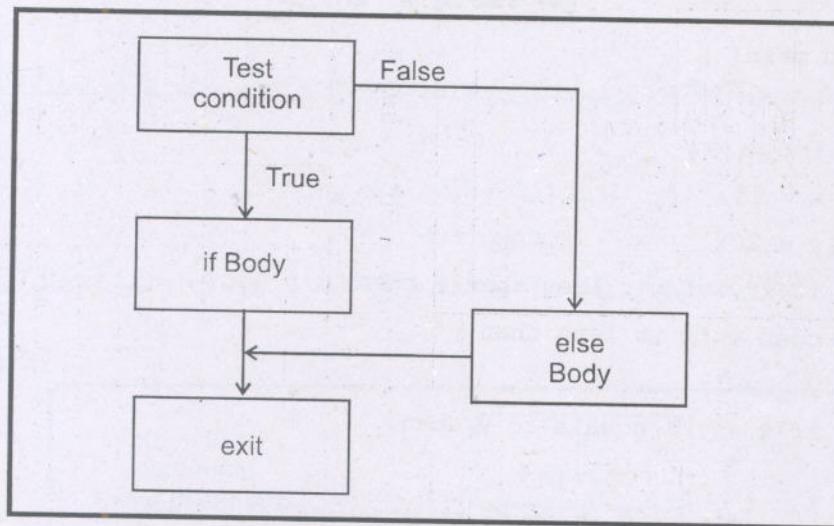
```
viod main( )
 {
 int x,y;
 x = 15;
 y = 20;
 if(x<y) //xis less than y then.
 cout <<"x is less than y";
 y = y-3;
 if(x==y) //equals to y then.
     {
     cout <<"x is equal to y" <<endl;
     cout <<"if statement example";
     }
 cout <<"\n End of Program.";
 getch( );
 }
```

اس پروگرام میں int ٹائپ کے دو ویری ایبلز x اور y ڈیکلیر اور انشلائز کروائے گئے ہیں اور ان کی مدد سے کنڈیشن چیک کی گئی ہے کہ اگر x ویری ایبل y سے چھوٹا ہے تو یہ ڈسپلے کرے اور بعد میں y میں سے 3 تفریق کیا ہے اور یہ کنڈیشن لگائی ہے کہ کیا x اور y دونوں برابر ہیں۔ اب اس پروگرام کو ایگزیکیوٹ کریں اور ایک بات کا خیال رکھیں کہ ہیڈر فائلز آپ خود لگائیں گے۔ اس کی آؤٹ پٹ یہ ہوگی۔

```
x is less than y
End of Program.
```

else-if سٹیٹ مینٹ:

آپ نے پیچھے if سٹیٹ مینٹ کے بارے میں پڑھا ہے اس کے ساتھ آپ else سٹیٹ مینٹ بھی استعمال کر سکتے ہیں جو اس کے فیچرز میں اضافہ

کرتی ہے۔اس کی مدد سے آپ if کنڈیشن کے علاوہ ایک اور متبادل کنڈیشن چیک کر سکتے ہیں۔آیئے اس کی حرآرکی سے اس کی اہمیت سمجھتے ہیں۔

```
if (condition)
  {
  statement;
  statement 1;
  }
  else
     statement;
```

else-if حرآرکی

آیئے اسی کی ایک سادہ اور آسان سی مثال لکھتے ہیں۔

مثال نمبر 2.2 else-if کا استعمال

```
viod main( )
  {
  int num1,num2;
  cout <<"Enter two integers.\n";
  cin >>num1 >>num2;
  if(num1%num2==0)
  cout <<"num1 <<"is divisable by" <<num2;
  else
  cout <<num <<"is not divisable by <<num2;
```

```
getch( );
}
```

اسی مثال میں ہم دونمریک نمبرز یوزر سے لے رہے ہیں اور پھر یہ کنڈیشن لگائی ہے کہ کیا پہلا نمبر دوسرے نمبر سے تقسیم ہو جاتا ہے یا نہیں۔

Nested-if سٹیٹ مینٹ:

آپ اپنے پروگرام میں جہاں ضرورت ہو if سٹیٹ مینٹ استعمال کر سکتے ہیں۔ اس طرح ایک if سٹیٹ مینٹ میں اور بھی if سٹیٹ مینٹس استعمال کی جاسکتی ہیں۔ اس کا جنرل طریقہ یہ ہے۔

```
if (condition)
  {
  if(condition)
     {
     if Body;
     }
  statement;
  }
  else
       statement;
```

Nested if سیٹ منٹ میں سب سے پہلے باہر والی if سٹیٹ مینٹ چیک کی جاتی ہے اگر وہ درست ہو تو پھر کنٹرول اس کے اندر لکھی ہوئی کنڈیشنل سٹیٹ مینٹ کو دے دیا جاتا ہے اور یوں یہ پراسس چلتا ہے۔ جب اس میں لکھی ہوئی سٹیٹ مینٹ ایگزیکیوٹ ہو جاتی ہے تو کنٹرول پر پہلی کنڈیشنل سٹیٹ مینٹ کوٹرانسفر کر دیا جاتا ہے۔ آیئے اس کی ایک آسان سی مثال دیکھتے ہیں۔

مثال نمبر 2.3 Nested-if سٹیٹ مینٹ

```
viod main( )
    {
    int var1,var2,var3,min;
    cout <<"Enter three numeric values:";
    cin >>var1 >>var2 >>var3;
    if(var1<var2) //var1 is less than var2
    if(var1<var3) //var1 is less than var3
    min=var1;      //min=var1
    else
    min=var3;      //min=var3
```

```
else
if(var2<var3) //var2 is less than var3
min=var2;       //min=var2
else
min=var3;       //min=var3
cout <<num <<"The minimum number is:" <<min;
}
```

اب فرض کریں کہ آپ مندرجہ ذیل ان پٹ دیتے ہیں۔

```
Enter Three values in Numbers: 212, 137, 41
The minimum number is: 41
```

اب اس کو سمجھنے کی کوشش کریں سب سے پہلے (var1<var2) غلط ہو گیا۔ اب ہماری دوسری کنڈیشن else ایگزیکیوٹ ہو گی اسی میں (var1<var2) کنڈیشن سے یہ بھی غلط ہو گی۔ اب سب سے آخری بلاک else ایگزیکیوٹ ہوگا اور سب سے کم نمبر سکرین پر ڈسپلے ہو جائے گا۔ اس کے بعد اگر آپ اس طرح ویلیوز درج کرتے ہیں۔

```
Enter Three values in Numbers: 41, 137, 212
The minimum number is: 41
```

اس کیس میں سب سے پہلے (var1<var2) کنڈیشن چیک ہوگی یہ درست ہے اس کے بعد دوسری کنڈیشن (var1<var3) ایگزیکیوٹ ہوگی وہ بھی درست ہے اس کے بعد والی سٹیٹ مینٹ ایگزیکیوٹ ہوگی جس میں var1 کی ویلیو min ویری ایبل کو آسائن کر دی جائے گی اور یوں سب سے چھوٹا نمبر حاصل ہو جائے گا۔

switch سٹیٹ مینٹ:

یہ سٹیٹ مینٹ if سٹیٹ مینٹ کی طرح کام کرتی ہے۔ فرق صرف اتنا ہے کہ یہ ایک سٹیٹ مینٹ میں ایکسپریشن کی کئی ویلیوز کو چیک کرنے سے سہولت فراہم کرتی ہے۔ بہر حال یہ if اور else if کا متبادل ہے۔ اس کو لکھنے کا جنرل طریقہ یہ ہے۔

```
Switch(expression)
  {
  Case value1;
      Statement;
      break;
  Case value2;
      break;
  :
  default:
```

```
        default statement;
    }
```

switch سٹیٹ مینٹ میں آپ کو کئی اہم باتوں کا خیال رکھنا ہوتا ہے کہ اس میں آپ کے وری ایبل کی ٹائپ یعنی ایکسپریشن کی ویلیو short, byte, int یا char میں ہونی چاہیے۔ اور ہر case ویلیو کو منفرد ہونا چاہیے۔ یہ ویلیوز مستقل ہوں یہ اعشاری نظام وغیرہ کو ہینڈل نہیں کرتیں۔ اس میں یہ ہوتا ہے کہ ایکسپریشن کو ہر case ویلیو کے ساتھ ملایا جاتا ہے۔ جس کی ویلیو برابر ہو اس کی سٹیٹ مینٹ ایگزیکیوٹ کر دی جاتی ہے اور اگر کوئی بھی case ویلیو ایکسپریشن سے نہ ملے تو پھر default (ڈیفالٹ) ویلیو پرنٹ کر دی جاتی ہے۔ اس کی مثال نیچے درج ہے۔

مثال نمبر 2.4 switch سٹیٹ مینٹ

```
viod main( )
    {
    clrscr( );
    int temp=40,temp2=30;
    int sum=temp+temp2;
    switch(sum)
        {
        case 80:
        cout <<"Your Grade is: A";
        break;
        case 70:
        cout <<"Your Grade is: B";
        break;
        case 60:
        cout <<"Your Grade is: C";
        break;
        default;
        cout <<"Your are Failed:";
        }
    getch( );
    }
```

جب آپ اس پروگرام کو ایگزیکیوٹ کریں گے تو یہ temp اور temp2 کو جمع کرے گا اور ان کا رزلٹ sum وری ایبل میں سٹور کرے گا جس کو ہم نے بعد میں switch( ) میں سے پاس کیا ہے اس پروگرام کی یہ آؤٹ پٹ یہ ہوگی۔

```
Your Grade is: B
```

آیئے switch کی ایک اور مثال دیکھتے ہیں۔

مثال نمبر 2.5

```
viod main( )
   {
   clrscr( );
   int a; char b;
   cout <<"Enter a No. between 1-8";
   cin >>a;
   switch(a)
      {
      case 1:
      case 2:
      case 5:
      case 7:
      b = 'odd';
      break;
      case 2:
      case 4:
      case 6:
      case 8:
      b = 'even';
      break;
   default:
      b='sorry':
      }
   cout <<"Your No. is" <<b;
   getch( );
   }
```

اس پروگرام میں ہم نے یوزر سے ان پٹ لی ہے کہ وہ اسے 9 کے درمیان کوئی بھی نمبر تحریر کرے۔ اب فرض کریں وہ اس سے بڑا نمبر لکھتا ہے تو پھر ڈیفالٹ سٹیٹ مینٹ پرنٹ ہوگی اور اگر وہ 1۔ سے 8 تک کوئی نمبر تحریر کرے گا تو پھر switch ایکسپریشن کو کیسس کی ویلیوز سے ملایا جائے گا اور اس کے مطابق رزلٹ ڈسپلے ہوگا۔ اس مثال کی مدد سے آپ سمجھ سکتے ہیں کہ ایک ایکسپریشن کو کس طرح زیادہ کیسز سے ملایا جاتا ہے۔

لاجیکل آپریٹرز:

لاجیکل آپریٹرز دو بولین (Boolean) ایکسپریشنز یا آپریٹرز کو ملانے کے لئے استعمال کئے جاتے ہیں۔ یعنی فرض کریں آپ کہتے ہیں کہ آج آپ نے چائے پی ہے؟ اب اس کے صرف دو ہی جواب ہیں ہاں یا نہیں۔ اب آپ کہتے ہیں کہ میں کھانا کھاؤں گا۔ ان کو آپ ملانا چاہتے ہیں کہ اگر آپ نے چائے پی ہے تو میں کھانا کھاؤں گا۔ تو یہاں (اور) کے طور پر استعمال ہو رہا ہے۔ ایسے حالات میں ہم لاجیکل آپریٹرز استعمال کرتے ہیں جب ہم دو کنڈیشنز کو ملانا چاہتے ہیں۔ یہ آپریٹرز مندرجہ ذیل ہیں۔

& &	And	(a<b && b>c) یہ اس وقت کام کرتا ہے جب دونوں کنڈیشنز درست ہوں۔
¦ ¦	OR	(a<b ¦ ¦ b>c) اس میں صرف ایک کنڈیشن کا درست ہونا ضروری ہے۔
!	Not	(a<b ! b>c) جب a غلط (0) ہو گا تو یہ کام کرے گا۔

اس کو ہم اس ٹرتھ ٹیبل سے سمجھتے ہیں۔

& &		
A	B	A&&B
1	1	1
0	1	0
1	0	0
0	0	0

¦ ¦		
A	B	A¦ ¦B
1	1	1
1	0	1
0	1	1
0	0	0

!	
A	B
1	0
0	1

اس میں آپ دیکھ رہے ہیں کہ && میں جہاں دونوں ویلیوز 1 ہیں وہاں رزلٹ 1 ہے جبکہ ¦¦ آپریٹر میں جہاں دونوں یا دونوں میں سے کوئی بھی ایک ویلیو 1 ہے تو رزلٹ 1 اور ! آپریٹر میں جب 0 ہے تو 1 اور جب 1 ہے تو 0 رزلٹ ہے۔ آیئے اس کی C++ میں ایک مثال دیکھتے ہیں۔

مثال نمبر 2.6 لاجیکل آپریٹرز کا استعمال

```
viod main( )
    {
    float average;
    int Maths,Physics,English, Sum;
    cout <<"Enter Marks of Three Subjects:";
    cin >>Maths >>Physics >>English;
    sum = Mahts+Physics+English;
    if(average==100 && average>90)
    cout <<"Your Grade is A+";
    else
    if(average<90 && average>80)
    cout <<"Your Grade is B+";
    else
```

```
if(average<80 && average>70)
cout <<"Your Grade is C+";
else
cout <<"You are failed";
getch( );
}
```

آیئے اب اس پروگرام کی ایگزیکیوشن ترتیب کو چیک کرتے ہیں۔ سب سے پہلے یہ تین ان پٹ یوزر سے مانگتا ہے جو کہ نمبرز ہوں گے۔ اس کے بعد ان کو جمع کر کے رزلٹ sum ویری ایبل میں سٹور کرتا ہے یا اس کی اوسط معلوم کر کے فلوٹ ٹائپ کے ویری ایبل average کو آسائن کر دیتا ہے۔ اس پر ہم نے پھر چیک یعنی if کی کنڈیشن لگائی ہے۔ فرض کریں آپ یہ ان پٹ دیتے ہیں۔

```
Enter Marks of Three Subjects;    70  83  91
Your Grade is B
```

اس میں وہ ان تینوں نمبروں کی اوسط معلوم کرنے کے بعد چیک کرے گا کہ اوسط کیا ہے اس صورت میں اوسط 80 سے زیادہ ہے۔ لہٰذا پہلی if کنڈیشن غلط ہو گئی اب دوسری چیک ہو گی وہ درست ہے۔ اس کے اندر موجود سٹیٹ مینٹ ایگزیکیوٹ ہو جائے گی اور اس کے بعد یہ پروگرام بند ہو جائے گا۔ اس پروگرام میں دونوں ایکسپریشنز چیک ہوں گی۔ اگر ان میں سے ایک بھی ایکسپریشن غلط ہو تو پھر وہ کنڈیشن غلط ہو جائے گی۔

آیئے اب ایک چھوٹا سا پروگرام لکھتے ہیں جس میں ||(OR) آپریٹر استعمال کیا گیا ہے۔

```
{
int a,b;
cout <<"Enter input";
cin >>a >>b;
if(a%b==0 || b%a=0)
cout <<"a and b are divisable";
}
```

اس پروگرام میں اگر دونوں میں سے ایک ایکسپریشن درست ہو گی تو آپ کی کنڈیشن درست تصور کی جائے گی اور اس کنڈیشن کی باڈی بھی ایگزیکیوٹ ہو جائے گی۔

ویری ایبل سکوپ:

کسی بھی ویری ایبل یا ایڈینٹیفائر کے سکوپ سے یہ مراد ہے کہ وہ پروگرام کے کسی حصے میں استعمال کیا جا سکتا ہے اور کون سا حصہ اسے استعمال نہیں کر سکتا۔ مثلاً آپ ایک ویری ایبل اس وقت تک پروگرام میں استعمال نہیں کر سکتے جب تک کہ وہ ڈیکلیر نہیں کیا جائے گا اور اس کا سکوپ وہی ہو گا جہاں یہ ڈیکلیر کیا گیا ہو۔ اس کی اہمیت کا اندازہ آپ کو آگے اس کتاب میں ہو گا۔ اس وقت اس کو صرف سمجھنے کی کوشش کریں۔

```
void main( )
{
int a=5;
int b=9;
```

```
{
a=11;  //You can use it here.
b=12;
c=3;   //Error it is not is scope.
int c;
a=2;
c=5;
}
b=3;    //Ok you can use it.
c=9;    //Error out of scope.
}
```

اس پروگرام میں ویری ایبل a اور b تمام پروگرام میں کہیں بھی استعمال کر سکتے ہیں جبکہ c کا سکوپ صرف اندر والی بریکٹیں ہیں۔اس کے علاوہ آپ اسے استعمال کریں گے تو یہ ایرر شو کرے گا۔آپ ایک پروگرام میں ایک ہی کام کے کئی ویری ایبل ڈیکلیر کر سکتے ہیں لیکن ان کا سکوپ مختلف ہونا چاہئے۔یہ کیسے ممکن ہے آیئے اس کی ایک مثال دیکھتے ہیں۔

مثال نمبر 2.7 Nested اور Parallel سکوپ

```
#include<iostream.h>
#include<conio.h>
int a=5;    //This is global variable.
void main( )
{
int a=9;
  {
  int a=101;
  cout <<"Inside the main inner block a=" <<a <<endl;
  }       //end of inner block
cout <<"In main( ) a=" <<a <<endl;
cout <<"global variable a=" <<::a <<endl;
getch( );
}       //end of main( )
```

اب اس پروگرام کو ایگزیکیوٹ کریں تو کچھ اس طرح کی آؤٹ پٹ ہوگی۔

```
Inside tha main inner block a=101
In main( ) a=9
global variable a=5
```

اسی پروگرام میں ایک ویری ایبل a کے مختلف تین اوبجیکٹس ہیں۔ پہلا a جس کی ویلیو 5 ہے وہ گلوبل ویری ایبل ہے۔ اس کو آپ تمام فائل میں کہیں بھی استعمال کر سکتے ہیں کیسے؟ یہ آپ فنکشنز کلاسز میں پڑھیں گے۔ اس کے بعد بھی a کی ویلیو 9 ہے وہ main( ) کے بلاک میں کہیں بھی استعمال کیا جا سکتا ہے۔ جبکہ 101 ویلیو کے ساتھ اینی شلائیز کیا ہوا ویری ایبل a صرف اسی بلاک میں استعمال کیا جا سکتا ہے۔ آپ نے جو a ویری ایبل main( ) میں ڈیکلیر کیا ہے وہ پہلے گلوبل ویری ایبل کا سکوپ چھپا دیتا ہے اور اسی طرح اندر والے بلاک میں a ویری ایبل پہلے دونوں ویری ایبلز کو چھپا دیتا ہے۔

اس طرح آپ جب بھی کسی چھپے ہوئے ویری ایبل کو ایکسس کرنا چاہتے ہیں تو اس کے لئے ریزرویشن آپریٹر (::) استعمال کیا جاتا ہے۔ جیسا کہ اوپر مثال نمبر 2.7 میں کیا گیا ہے۔

لوپس:

ایک لوپ کی مدد سے آپ اپنے پروگرام کا ایک مخصوص حصہ بار بار (کئی دفعہ) ایگزیکیوٹ کروا سکتے ہیں۔ آپ کا یہ حصہ اس وقت تک ایگزیکیوٹ ہوتا رہتا ہے جب تک کنڈیشن درست ہوگی۔ جونہی کنڈیشن غلط ہو جائے گی تو یہ لوپ ختم ہو جائے گا اور لوپ سے باہر ٹرانسفر کر دیا جائے گا۔ C++ میں لوپس کی تین اقسام ہیں۔

(i) For لوپ

(ii) While لوپ

(iii) Do-while لوپ

for لوپ:

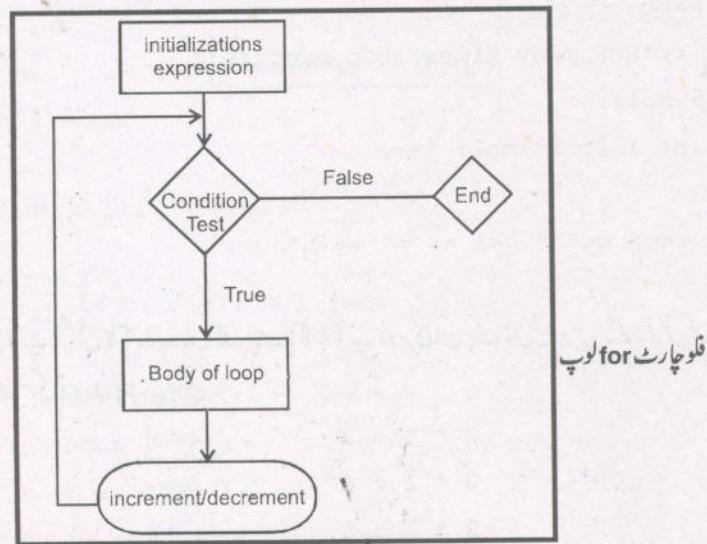
for لوپ آپ کے پروگرام کے مخصوص کوڈ کو محدود وقت کے لئے ایگزیکیوٹ کرتا ہے۔ یعنی یہ لوپ آپ اس وقت استعمال کرتے ہیں جب آپ کو معلوم ہو کہ یہ کوڈ اتنی دفعہ ایگزیکیوٹ ہونا چاہئے۔ ہر لوپ کے تین حصے ہوتے ہیں۔

(i) لوپ ویری ایبل اینی شلائیز کرنا

(ii) لوپ کنڈیشن یا لوپ ختم کرنے کے لئے ویلیو دینا جسے Terminating ویلیو بھی کہتے ہیں

(iii) ویری ایبل اپ ڈیٹ (Increment / Decrement) کرنا

for لوپ کا جنرل طریقہ یہ ہے۔

```
for(initialization; condition; increment/decrement)
    {
        Body;
    }
```

اگر آپ کے for لوپ میں صرف ایک ہی سٹیٹ مینٹ ہے تو پھر { } (بریکٹس) کی کوئی ضرورت نہیں ہوتی۔ لوپ for( ) میں تین حصے ہیں جو ایک دوسرے سے سیمی کالن (;) کے ذریعے جدا جدا کئے گئے ہیں۔ جب پہلی دفعہ لوپ شروع ہوگا تو اینی شلائیزیشن والا حصہ ایگزیکیوٹ ہوتا ہے۔ یہ لوپ کے ویری ایبل کی ویلیو سیٹ کرنے کے لئے استعمال ہوتا ہے اور سب سے اہم بات یہ ہے کہ یہ پورشن لوپ کی زندگی میں صرف ایک بار ایگزیکیوٹ ہوتا ہے۔ اگر یہ ہر بار ایگزیکیوٹ ہوتو پھر ہر دفعہ ویری ایبل کی ویلیو ہی سیٹ ہو جائے اس کے بعد کنڈیشن چیک ہوتی ہے اور یہ ہر دفعہ چیک ہوگی یہ بولین ٹائپ کی کنڈیشن ہوتی ہے یعنی درست یا غلط۔ اگر یہ کنڈیشن درست ہوگی تو لوپ کی باڈی ایگزیکیوٹ ہوگی ورنہ لوپ ختم ہو جائے گی۔ اس کے بعد تیسرا حصہ

اپ ڈیٹ کا ہوتا ہے اسے Iteration بھی کہتے ہیں۔ یہ لوپ ویری ایبل کی ویلیو کو اپ ڈیٹ کرتا ہے۔ for لوپ کا فلو چارٹ یہ ہوگا۔

فلو چارٹ for لوپ

آیئے اب اس کا ایک پروگرام لکھتے ہیں جو ٹیبل پرنٹ کروائے۔

مثال نمبر 2.8 for لوپ پروگرام

```
void main( )
    {
    int no,ans;
    cout <<"Enter the Table Number";
    cin >>no;
    for(int i=1;i<=10;i++)
        {       //start for loop
        ans i*no;
        cout <<no <<"*" <<i <<"=" <<ans <<endl;
        }       //End for loop
    getch( );
    }       //end main( )
```

اب آپ جب اس پروگرام کو ایگزیکیوٹ کریں گے تو سب سے پہلے یہ ایک ان پٹ نمبر مانگے گا کہ کس نمبر کا آپ ٹیبل معلوم کرنا چاہتے ہیں۔ اس کے لئے مطلوبہ نمبر تحریر کریں۔ اس کا ٹیبل پرنٹ ہو جائے گا۔

اس کے علاوہ آپ ایک اور کام یہ بھی کر سکتے ہیں کہ یوزر کو کنٹرول دیں یعنی لوپ کتنی دفعہ چلے تو اس کے لئے دو ان پٹس لیں اور کنڈیشن میں دوسری ان پٹ استعمال کریں۔ مثلاً

```
    int no, ans, no1;
```

```
cout <<"Enter Table Number:"
cin >>no;
cout <<"How many times loop execute";
cin >>no1;
for(int i=1; i<=no1; i++)
{
cout <<no <<"*" <<i <<"=" <<(i*no);
}
```

اب اس پروگرام میں لوپ یا ٹیبل کا آرڈر وہاں تک جائے گا جو آپ دوسری ان پٹ میں نمبر دیں گے۔ فرض کریں آپ پہلے 2 اور بعد میں 3 ان پٹ دیتے ہیں تو اس پروگرام کی آؤٹ پٹ یہ ہوگی۔

```
2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
```

اسی طرح آیئے ایک پروگرام لکھتے ہیں جس میں یوزر ان پٹ دیتا ہے اور ہم نے اس کا فیکٹر نمبر (factorial) معلوم کرنا ہے۔

مثال نمبر 2.9 فیکٹر نمبر معلوم کرنا

```
void main( )
{
int no,factor=1;
cout <<"Enter a +ve Number:";
cin >>no;
for(int i=2; i<=no;i++)
    factor *=i;
cout <<no <<"factorial is=" <<factor;
getch( );
}
```

اس پروگرام میں آپ جو مثبت نمبر تحریر کریں گے اس کا فیکٹر یوں معلوم کرے گا فرض کریں آپ 5 نمبر ان پٹ کے طور پر دیتے ہیں تو یہ سب سے پہلے i کو factor سے ضرب دے گا اور رزلٹ اس میں سٹور کر دے گا اور یہ اس وقت تک ان دونوں ویری ایبلز کو ضرب دیتا رہے گا جب تک کنڈیشن غلط نہیں ہو جائے گی۔ یعنی i کی ویلیو 5(no==5) نہیں ہو جائے گی پھر بعد میں اس کی آؤٹ پٹ یہ دے گا۔

```
5 factorial is = 120
```

Nested-for لوپ:

دوسری پروگرامنگ لینگوئجز کی طرح C++ بھی Nested For لوپ کی سہولت فراہم کرتی ہے۔ اس کا طریقہ بالکل if کنڈیشن کی طرح ہے۔ اس سے for لوپ کے فیچرز میں اضافہ ہوتا ہے۔ جب آپ Nested for لوپ استعمال کرتے ہیں تو اس میں سب سے پہلے باہر والی لوپ ایگزیکیوٹ ہوگی

اس کے بعد اس کے اندر والی لوپ چلے گی اگر باہر والی لوپ کی کنڈیشن درست ہوگی۔ اس کے بعد اگر لوپ کی کنڈیشن درست ہوگی تو اس کی باڈی ایگزیکیوٹ ہوگی۔ آیئے اب اس کا ایک پروگرام لکھتے ہیں۔

مثال نمبر 2.10 Nested-for لوپ

```
void main( )
    {
    for(int i=1;i<=4;i++)
        {
            for(int j=1;j<=i;j++)
            {
            for(int k=1;k<=j;k++)
            cout <<"*";
            cout <<endl;
            }   //end of inner loop
        cout <<endl;
        }   //end of outer loop
    getch( );
    }       //end main( )
```

اس پروگرام میں سب سے پہلے باہر والا لوپ ایگزیکیوٹ ہوگا اس کے بعد دوسرا لوپ چلے گا پھر تیسرا لوپ چلے گا اور اس میں جو سٹیٹ مینٹ ہے وہ پرنٹ ہوگی۔ یہ لوپ چار دفعہ چلے گا اس کے بعد کنٹرول دوسرے (inner) لوپ کو ٹرانسفر کر دیا جائے گا۔ اس طرح یہ لوپ اس وقت تک چلے گی جب تک اس کی کنڈیشن درست رہے گی۔ پھر اس کے بعد کنٹرول پہلے (outer) لوپ کو ٹرانسفر کر دیا جائے گا اور یوں یہ پراسیس اس وقت تک چلتا رہے گا جب تک outer (باہر والے) لوپ کی کنڈیشن درست رہے گی۔ اس پروگرام کی آؤٹ پٹ یہ ہوگی۔

```
*
* *

*
* *
* * *

*
* *
* * *
* * * *
```

```
*
* *
* * *
* * * *
* * * * *
```

while لوپ:

while لوپ بھی for لوپ کی طرف ایک مخصوص کوڈ کو بار بار ایگزیکیوٹ کرنے کے لئے استعمال ہوتی ہے لیکن یہ اس وقت استعمال کی جاتی ہے جب ہمیں یہ کنفرم نہیں ہوتا کہ لوپ کی باڈی کتنی دفعہ ایگزیکیوٹ ہونی چاہیے۔ اس کا جنرل طریقہ کار یہ ہے۔

```
while (condition)
    {    //body of loop;
    }
```

اس میں سب سے پہلے کنڈیشن چیک کی جاتی ہے۔ اگر یہ درست ہوگی تو لوپ کی باڈی ایگزیکیوٹ ہوگی اور اس کے بعد پھر کنڈیشن چیک کی جائے گی۔ یہ سلسلہ اس وقت تک جاری رہتا ہے جب تک کنڈیشن درست رہے گی۔ جونہی کنڈیشن غلط ہو جائے گی کنٹرول لوپ کے باہر ٹرانسفر کر دی جائے گی۔ نیچے اس کی ایک مثال ہے۔ جس میں الٹ آرڈر میں چند نمبر ڈسپلے کروائے گئے ہیں۔

مثال نمبر 2.11 while loop کا استعمال

```
void main( )
    {
    int i=4;
    while(i>=0)     {   //start of loop
        cout <<i <<"/n";
        i--;            //decrement to control loop
        }   //end of while loop
    getch( );
    }               //end main( )
```

اس پروگرام میں i ویری ایبل کی ویلیو 4 ہے اور لوپ میں (i>=0) لکھا ہے یعنی جب تک i کی ویلیو 0 کے برابر یا زیادہ ہوگی لوپ کی باڈی ایگزیکیوٹ ہوگی اور باڈی میں i کی ویلیو پرنٹ کی جا رہی ہے اور ساتھ ہی اس میں ایک کی کمی بھی کی جا رہی ہے تا کہ while کی لوپ کی کنڈیشن غلط بھی ہو اور پروگرام exit ہو جائے۔ اس پروگرام کی آؤٹ پٹ یہ ہوگی۔

```
4
3
2
1
0
```

فرض کریں آپ کے پاس ایک ویری ایبل i=3 ہے اور آپ while میں یہ ایکسپریشن لکھتے ہیں۔

```
while(i>5) {
    cout <<"Hello";
    i++;
    }
```

تو یہ پروگرام کوئی آؤٹ پٹ شو نہیں کرے گا کیونکہ آپ کے پاس i کی ویلیو 3 ہے۔ جب ہم کنڈیشن میں یہ چیک کر رہے ہیں کہ جب i>5 (i پانچ سے بڑا ہو) تب لوپ سٹیٹ مینٹ چلے جو کہ غلط ہے۔ اس لئے پروگرام کوئی آؤٹ پٹ ڈسپلے نہیں کرے گا۔

do-while لوپ:

آپ نے اوپر دیکھا کہ while لوپ میں سب سے پہلے کنڈیشن چیک کی جاتی ہے اور اگر پہلی دفعہ ہی کنڈیشن غلط ہو تو لوپ سٹیٹ مینٹ ایگزیکیوٹ نہیں ہوتی۔ لیکن بعض اوقات ایسی صورت حال پیش آجاتی ہے کہ آپ چاہتے ہیں کم از کم ایک دفعہ ہماری لوپ باڈی ایگزیکیوٹ ضرور ہو خواہ کنڈیشن درست ہو یا غلط۔ تو اس کے لئے آپ do-while لوپ استعمال کر سکتے ہیں۔ اس کا جنرل طریقہ یہ ہے۔

```
do  {
        loop Body;
    }   while(condition);
```

اس میں لوپ باڈی کم از کم ایک دفعہ اس لئے ایگزیکیوٹ ہوتی ہے کہ اس میں پہلے لوپ کی باڈی ایگزیکیوٹ ہوتی ہے اور اس کے بعد کنڈیشن چیک کی جاتی ہے اور اگر کنڈیشن درست ہوگی تو دوبارہ کنٹرول لوپ میں ٹرانسفر ہوگی ورنہ لوپ ختم ہو جائے گا۔ آیئے اس کی مدد سے ایک پروگرام لکھتے ہیں اور اس کی آفادیت سمجھتے ہیں۔

مثال نمبر 2.12 do-while لوپ

```
void main(void)
   {
   clrscr( );
   int temp=1,temp2=1,end;
   cout <<"How many times loop is executed:";
   cin >>end;
   do
       {
       temp*=temp2;
       temp2++;
       }
       while(temp2<=end);
   cout "\n Answer is:" <<temp;
   getch( );
   }
```

اس پروگرام میں do-while لوپ استعمال کیا گیا ہے۔ یہ پروگرام ایک یوزر سے ان پٹ لے رہا ہے اور آپ جو نمبر درج کریں گے اتنی دفعہ یہ لوپ ایگزیکیوٹ ہوگی اور اصل میں یہ اس نمبر کا فیکٹر نمبر معلوم کرے گا۔ آیئے اس کی آؤٹ پٹ دیکھتے ہیں۔

```
How many times loop is executed : 5
Answer is : 120
```

بریک سٹیٹ مینٹ:

بریک سٹیٹ مینٹ کی مدد سے آپ کسی بھی سٹیٹ مینٹ کی ایگزیکیوشن ختم کر سکتے ہیں۔ جیسا کہ آپ نے اس سے پہلے switch سٹیٹ مینٹ میں دیکھا ہے۔ آپ یہ سٹیٹ مینٹ لوپ میں بھی استعمال کر سکتے ہیں اس سے لوپ کنٹرول باہر دوسری سٹیٹ مینٹس کو ٹرانسفر کر دیا جائے گا۔ اب جہاں تک اس ضرورت کا تعلق ہے یعنی یہ کہاں کہاں استعمال کی جا سکتی ہے تو وہ مختلف پرابلمز پر منحصر ہے اس کو آپ C++ میں کیسے استعمال کر سکتے ہیں۔ آیئے اس کیلئے ایک مثال سے مدد حاصل کرتے ہیں۔

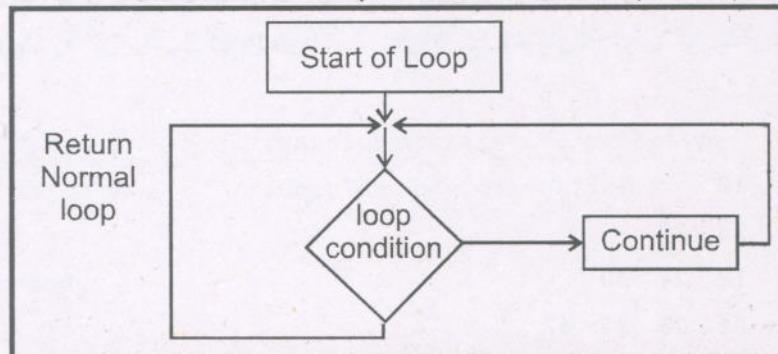
مثال نمبر 2.13 break سٹیٹ مینٹ کا استعمال

```
void main( ){
    for(int i=1; i<=7; i++)    {
        if(i==4)
        break;        //terminate loop
        cout <<"i=" <<i <<endl;
        }             //end of loop
    cout <<"loop is terminated";
    getch( );
    }       //end of main
```

اس پروگرام میں لوپ کی ویلیو 1 سے 7 تک سیٹ کی ہے اور لوپ میں if کی کنڈیشن استعمال کی گئی ہے کہ جب i==4 ہو جائے تو لوپ ختم ہو جائے جب تک i کی ویلیو 4 نہیں ہوگی لوپ کی باڈی ایگزیکیوٹ ہوتی رہے گی۔ جونہی i==4 ہو جائے گی لوپ سے کنٹرول باہر والی سٹیٹ مینٹ کو ٹرانسفر کر دیا جائے گا۔ یوں اس پروگرام کی آؤٹ پٹ یہ ہوگی۔

```
i = 1
i = 2
i = 3
Loop is terminated
```

continue سٹیٹ مینٹ:

یہ سٹیٹ مینٹ بریک سٹیٹ مینٹ کے متضاد ہے۔ جب آپ اس کو استعمال کرتے ہیں تو آپ کے پروگرام کا وہ حصہ ایگزیکیوٹ ہوتا رہتا ہے۔ یعنی اگر آپ نے لوپ میں یہ سٹیٹ مینٹ استعمال کی ہے تو یہ اسے کچھ دیر کے لئے نظر انداز کر دے گا۔ مثلاً

```
for(int i=1; i<=20; i++)   {
    if(i%2==0)
```

```
continue;
sum+=i;      }
```

یہ پروگرام کس طرح کام کرے گا آپ continue سٹیٹ مینٹ کے فلو چارٹ سے سمجھ سکتے ہیں۔

goto سٹیٹ مینٹ:

آپ نے اس سے پہلے بریک، سوئچ اور (continue) کنٹی نیوسٹیٹ مینٹ کے بارے میں پڑھا ہے۔ یہ تمام سٹیٹ مینٹس پروگرام کا کنٹرول واپس اسی جگہ ٹرانسفر کرتی ہیں جہاں سے یہ عام طور پر ٹرانسفر کیا جاتا ہے۔ continue سٹیٹ مینٹ میں کنٹرول واپس لوپ کنڈیشن کو ٹرانسفر کر دیا جاتا ہے۔ جبکہ switch سٹیٹ مینٹ میں کنٹرول درست سٹیٹ مینٹ پر جاتا ہے۔ ان تمام سٹیٹ مینٹس کو جمپ (jump) سٹیٹ مینٹس کہتے ہیں۔

اس کے علاوہ jump سٹیٹ مینٹ کی ایک اور قسم بھی ہے جس کو goto سٹیٹ مینٹ کہتے ہیں۔ اس میں آپ خود سے لیبل لگاتے ہیں کہ پروگرام کا کنٹرول کہاں ٹرانسفر کیا جائے۔ لیبل ایک ایڈینٹیفائر کی طرح ہوتا ہے جس کے بعد کالن (:) لگا ہوتا ہے یہ آپ کہیں بھی لگا سکتے ہیں۔ یہ سٹیٹ مینٹ C++ لینگوئج میں کیسے استعمال کی جاتی ہے آیئے اس کی ایک مثال دیکھتے ہیں۔

مثال نمبر 2.14 Goto سٹیٹ مینٹ

```
void main( )    {
    outer:
    for(int i=1; i<10; i++)    {
        for(int j=1; j<i; j++)
            {
            if(j>i)
            goto outer;
            cout <<(i*j) <<" ";
            }  //end of inner loop
        cout <<endl;
    }  //end of outer loop
    getch( );
    }  //end of main
```

اس پروگرام میں ہم نے ایک لیبل :outer کے نام سے ڈیکلیر کیا ہے۔اس کے بعد دولوپس استعمال کئے ہیں۔یعنی Nested لوپس ان میں سے inner لوپ میں ایک if کی کنڈیشن لگائی ہے کہ اگر (j>i) ہے تو کنٹرول یہاں سے باہر شروع میں پہلے لوپ (outer) کوٹرانسفر کر دیا جائے۔اگر ایسا نہیں ہوگا تو اس کنڈیشن کے بعد والی سٹیٹ مینٹ ایگزیکیوٹ ہوگی اور پروگرام نارمل ایگزیکیوٹ ہوگا۔اس پروگرام کو جب آپ ایگزیکیوٹ کریں گے تو یہ آؤٹ پٹ ڈسپلے ہوگی۔

```
2
3   6
4   8   12
5   10  15  20
6   12  18  24  30
7   14  21  28  35  42
8   16  24  32  40  48  56
9   18  27  36  45  54  63  72
```

کانسٹنٹ اینڈ اوبجیکٹس: (Constants and Objects):

اوبجیکٹ ایک میموری کا حصہ ہوتا ہے جس کا ایک ایڈریس، سائز، ٹائپ اور ویلیو ہوتی ہے۔اوبجیکٹ کا ایڈریس اصل میں میموری کی پہلی بائٹ کا ایڈریس ہوتا ہے۔اوبجیکٹ کے سائز سے مراد میموری میں اس کا سائز ہوتا ہے یعنی اس کے لئے کئی بائیٹس ریزرو ہیں۔ٹائپ میں اوبجیکٹ میں سٹور ویلیو کی ٹائپ ہوتی ہے اور ویلیو وہ کانسٹنٹ ہوگا جو میموری میں محفوظ کیا جائے گا۔اوبجیکٹ کی ٹائپ پروگرام واضح کرتا ہے اور اس کی ویلیو پروگرام خود سے کمپائل ٹائم پر سیٹ کرتا ہے یا رن ٹائم پر یہ آپشن یوزر کو بھی دی جاسکتی ہے۔آپ نے ویری ایبل کے بارے میں تو پڑھا ہے کہ جس کی ویلیو تبدیل ہوتی رہتی ہے۔جب کہ ایسا اوبجیکٹ جس کی ویلیو تبدیل نہ ہو کانسٹنٹ (مستقل) کہلاتا ہے۔مستقل اوبجیکٹ ڈیکلیر کرنے کے لئے ساتھ یہ واضح کرنا ہوتا ہے کہ یہ ویلیو مستقل ہے اور تبدیل نہیں کی جاسکتی اس کے لئے کی ورڈ const استعمال ہوتا ہے۔مثلاً

```
const in a=11;
```

اور ویری ایبلز کے برعکس کانسٹنٹس جب ڈیکلیر کئے جاتے ہیں تو اس وقت انی شلائیز کرنا بھی ضروری ہے۔آیئے اس کی ایک مثال دیکھتے ہیں۔

```
void main( )
    {
    const char beep='\b';
    const float PI=3.14;
    float n=PI/2;
    }
```

فنکشنز:

ایسے پروگرامز جن میں حقیقی پرابلمز حل کئے ہوتے ہیں اور یہ پرابلمز بہت بڑے بھی ہو سکتے ہیں۔ہم نے اب تک جو بھی پروگرام بنائے ہیں وہ صرف بنیادی پروگرام ہیں۔اور کسی بھی اہم پرابلم کو حل کرنے کے لئے آپ کا پروگرام جتنا بڑا ہوگا اتنا ہی مشکل ہوگا اس کو سمجھنا اور اپ ڈیٹ یا ایڈٹ کرنا بھی اتنا ہی زیادہ مشکل ہوتا ہے۔بڑے پروگرام کو بامعانی، سادہ اور آسان بنانے کے لئے پروگرامرز اس کو چھوٹے چھوٹے سب پروگراموں میں تقسیم کر

لیتے ہیں۔ یہ سب پروگرامز فنکشن کہلاتے ہیں۔
C++ میں پہلے سے بنے ہوئے فنکشنز بھی ہیں۔ یہ C++ کی سٹینڈرڈ لائبریری میں سٹور ہیں۔ اس میں سٹرنگ، میتھس (Maths) اور کریکٹر سے متعلق کئی اہم فنکشنز ہیں جن کو پروگرامر استعمال کرسکتا ہے۔ یہ اصل میں پروگرامر کا کام آسان بنا دیتے ہیں۔ پروگرامر کسی مخصوص کوڈ کے لئے ایک فنکشن لکھتے ہیں جو ایک پروگرام میں مختلف پوائنٹس پر استعمال کیا جاتا ہے۔ فنکشن کال میں فنکشن کا نام ہوتا ہے۔ کسی بھی فنکشن کے لئے تین اہم پوائنٹس اس طرح لکھے جاتے ہیں۔

```
void func( );          //function declaration/prototype
void func( );          //function body/defination
    {
    //body
    }
    fun( );       //function calling
```

سٹینڈرڈ میتھس فنکشن:

میتھ فنکشنز کی مدد سے پروگرام بنیادی حسابی کیلکولیشنز پرفارم کرسکتے ہیں یعنی جذر، پاور اور لاگ وغیرہ معلوم کرنا۔ یہ C++ کی سٹینڈرڈ لائبریری میں ڈیفائن ہوتے ہیں۔ نیچے ہم نے ایک ٹیبل بنایا ہے جس میں میتھس کے اہم فنکشنز کی ایک فہرست ہے۔

میتھڈ (Method)	وضاحت	مثال
ceil(x)	یہ x نمبر کو راؤنڈ کرتا ہے یعنی اسے نزدیک ترین نمبر میں کنورٹ کرتا ہے	ceil(7.42) کو 8.0 میں کنورٹ کردے گا
cos(x)	یہ ٹرگنومیٹری میں Cosine کی ویلیو معلوم کرتا ہے	Cos(0) کی ویلیو 1 ہے
exp(x)	یہ پاور کے لئے استعمال ہوتا ہے e^x	exp(2) ریٹرنز 7.289
fabs(x)	یہ x نمبر کی مکمل ویلیو ریٹرن کرتا ہے	fabs(-2) ریٹرنز 2.0
floor(x)	یہ x نمبر کو راؤنڈ ڈاؤن کرتا ہے یعنی اعشاری نظام کرتا ہے	floor(3.2) کو 3 میں کنورٹ کرتا ہے
fmod(xy)	اس میں دو ویلیوز دی جاتی ہیں اور یہ x,y کا بقایا اعشاری نظام میں ظاہر کرتا ہے مثلاً x%y	fmod(4.126,2,19) اس کا بقایا یہ 1.907 ہوگا
log(x)	اس سے x کا لاگرتھم معلوم ہوگا قدرتی بیس e ہوتی ہے	log(3.276) کا جواب 0.515 ہے
log10(x)	x کا لاگرتھم جس کی بیس 10 ہوگی	log(100.0) کا جواب 2.0 ہے
pow(x,y)	اس کو کہتے ہیں کہ x کی طاقت y ہے (x^y)	Pow(2,5) کا جواب 32 ہے
sqrt(x)	اس سے مراد x نمبر کا جذر ہے	sqrt(81) کا جواب 9 ہے
sin(x)	اس سے آپ Sine کی ویلیو معلوم کرسکتے ہیں	sin(1) کا مطلب 0.0174 ہے
tan(x)	اس آپ ٹرگنومیٹری tan کی ویلیو معلوم کرسکتے ہیں	tan(30) کا مطلب 0.577 ہے

اس میں آپ نے دیکھا کہ زیادہ تر میتھڈز میں ان کے نام کے بعد بریکٹس (x) میں لکھا ہوا ہے۔ اس کا مطلب ہے آپ یہاں پر ویلیو درج کریں جس کا آپ رزلٹ معلوم کرنا چاہتے ہیں۔ اس کو تفصیل سے آپ بعد میں پڑھیں گے۔ اس کے علاوہ کچھ میتھڈز میں دو ویلیوز درج کرنے کا کہا گیا ہے۔

مثلاً pow(x,y) اس کا مطلب ہے کہ آپ اس میں سے دو آرگومینٹس پاس کر سکتے ہیں یعنی یہاں پر آپ کو دو ویلیوز درج کرنا ہوں گی۔ ان میتھڈز کا یہ فائدہ ہے کہ آپ اگر کسی نمبر کا جذر معلوم کرنا چاہتے ہیں یا لاگرتھم معلوم کرنا چاہتے ہیں تو اس سے متعلقہ فیلڈ کا کام لکھیں گے اور مطلوبہ ویلیوز درج کریں گے ویسے اگر آپ یہ فیلڈ استعمال نہیں کرتے تو لاگرتھم معلوم کرنے کے لئے آپ کو ایک لمبا کوڈ لکھنا ہوگا اور اس کے لئے آپ کو لاگرتھم نکالنے کا اصول بھی آنا چاہئے جو کافی مشکل ہے۔ یوں یہ پروگرامرز کی سہولت کے لئے ہے۔ C++ کی لائبریری math.h میں پہلے سے بنائے ہوئے میتھڈز ہیں۔ ان کو استعمال کرنا بہت آسان ہے اس کے لئے math.h ہیڈر فائل شامل کریں اور فنکشن کا کام اور ویلیو درج کریں۔ آئیے اس کی ایک مثال دیکھتے ہیں۔

مثال نمبر 2.15 پری ڈیفائن حسابی میتھڈز کا استعمال

```
#include <iostream.h>
#include <conio.h>
#include <math.h>
void main(void)
  {
  double num, ans;
  cout <<"Enter number to find square root";
  cin >>num;
  ans = sqrt(num);    //Calculate square root
  cout <<"The square root of" <<num <<"=" <<ans;
  getch( );
  }        //end of main
```

اس پروگرام میں سب سے پہلے دیکھیں ہم نے void main(void) لکھا ہے اس کا مطلب ہے کہ main میتھڈ نہ تو کوئی ویلیو ریٹرن کر رہا ہے اور نہ ہی کوئی آرگومینٹ پاس کر رہا ہے۔ اس کے بعد ایک یوزر سے ان پٹ لی گئی ہے جس کا جذر معلوم کرنا ہے اور بعد میں sqrt(num) میں یہ ویلیو پاس کی ہے اور اس کا رزلٹ ans کے ویری ایبل میں سٹور کیا ہے۔ آئیے اس کی ایک اور مثال دیکھتے ہیں۔
اب ہم ٹرگنومیٹری پرابلم sin2x=2sinxcosx حل کریں گے۔ اس کی ہیڈر فائلز آپ خود لکھیں گے۔ اس کا ہیڈر ہم یوں لکھیں گے۔

مثال نمبر 2.16 ٹرگنومیٹری پرابلم sin2x=2sinxcosx حل کرنا

```
void main(void)
 {
 cout <<setw(3) <<"x" <<setw(7) <<"sin2x" <<setw(10) <<"2sinxcosx";
 for (float x=0; x<2; x+=0.3)
 cout <<setw(3) <<x <<xsetw(7) <<sin(2*x)
 <<setw(10) <<(2*sin(x)*cos(x)) <<endl;
 }
```

اس پروگرام میں ایک لوپ استعمال کیا ہے جو 2 سے کم رہے گا جونہی ویلیو 2 یا 2 سے زائد ہوگی تو لوپ ختم ہو جائے گی۔ اس میں لوپ کے اضافے پر غور کریں ہم نے 0.3 کا اضافہ کیا ہے یعنی ضروری نہیں آپ کمی یا بیشی صرف 1 کی کریں۔ اپنی ضرورت کے مطابق آپ یہ کمی بیشی کر سکتے ہیں۔ اس

کے بعد sin(2*x) کو پرنٹ کروایا ہے جو sin2x کی ویلیو اصل میں پرنٹ کرتا ہے اس کے بعد (2+sin(x)*cos(x)) کی ویلیو پرنٹ کی گئی ہے اور 2sinxcosx کی ویلیو پرنٹ کروائی گئی ہے۔اس کی آؤٹ پٹ کچھ یوں ہوگی۔

```
x        sin2x        2sinxcos
0        0            0
0.3      0.564642     0.564642
0.6      0.932039     0.932039
0.9      0.973838     0.973838
1.2      0.675463     0.675463
1.5      0.14112      0.14112
1.8      -0.44252     -0.44252
```

یوزر ڈیفائن فنکشنز:

ہم نے پہلے بھی بتایا ہے کہ تمام فنکشنز پروگرام کو چھوٹے چھوٹے سب پروگرامز یا موڈ لورائیز (Modularize) کرنے کے لئے استعمال ہوتے ہیں۔ایسے تمام ویری ایبلز جو فنکشنز کی ڈیفینیشن یعنی باڈی میں ڈیکلیر کئے جاتے ہیں لوکل ویری ایبل کہلاتے ہیں۔ویری ایبلز دو قسم کے ہوتے ہیں لوکل ویری ایبل اور گلوبل ویری ایبل۔ان کا تعارف آپ سکوپ میں پڑھ چکے ہیں۔لوکل ویری ایبلز صرف اسی ایریا یعنی { } میں استعمال ہو سکتے ہیں جن میں وہ ڈیکلیر کئے گئے ہوں۔فنکشنز بنانے کا ایک اہم فائدہ یہ ہے کہ آپ اسے ایک پروگرام میں کہیں بھی صرف نام لکھ کر استعمال کر سکتے ہیں۔یعنی پورا کوڈ لکھنے کی ضرورت نہیں ہوتی صرف فنکشن کال کریں۔

یوزر ڈیفائن فنکشن کے دو حصے ہوتے ہیں۔اس کا ہیڈر اور باڈی۔ ہیڈر میں فنکشن کا نام، ریٹرن ٹائپ اور پیرامیٹر کی فہرست ہوتی ہے۔فنکشن کی باڈی { } کے درمیان لکھی جاتی ہے اور یہ فنکشن کے ہیڈر کی پیروی کرتی ہے۔اس میں وہ کوڈ لکھا جاتا ہے جو کوئی ایکشن پرفارم کرتا ہے اور ریٹرن ویلیو بھی اسی حصہ میں سیٹ کی جاتی ہے۔آیئے یوزر ڈیفائن فنکشن کی ایک مثال دیکھتے ہیں۔

مثال نمبر 2.17 یوزر ڈیفائن سادہ فنکشن

```
#include <iostream.h>
#include <conio.h>
void func(void);              //function prototype
void main(void)
    {
    clrscr( );
    cout <<"Main Program" <<endl;
    cout <<"Here is after function calling" <<endl;
    func( );                  //function calling
    getch( );
      }
```

```
void func(void)    //function defination
{
float rad, result;
const float PI=3.14;
cout <<"Enter radius:";
cin >>rad;
result = rad×rad×PI;
cout <<"\n Answer is:" <<result:
}
```

اس پروگرام کی آؤٹ پٹ یہ ہوگی۔

```
Main Program
Here is after function calling
Eneter radius : 4.12
Answer is : 53.299614
```

اس پروگرام میں ہم نے ایک سادہ فنکشن بنایا ہے یعنی یہ فنکشن نہ تو ویلیو ریٹرن کرتا ہے اور نہ ہی اس میں سے کوئی آرگومنٹ پاس کیا جارہا ہے۔ ہم نے فنکشن ( )function میں ایک کانسٹنٹ فلوٹ ویری ایبل PI ڈیکلیر کیا ہے اور دو ویری ایبلز فلوٹ ٹائپ کے rad اور result ڈیکلیر کئے ہیں۔ اس میں rad کی ویلیو یوزر سے لی جارہی ہے جبکہ result ویری ایبل میں ایکسپریشن کا جواب سٹور کروایا گیا ہے جو بعد میں پرنٹ کروایا گیا ہے۔

ریٹرن ٹائپ فنکشنز:

ایسے فنکشنز جن میں سے آپ کوئی ویلیو ریٹرن کرنا چاہتے ہیں اس کی ریٹرن ٹائپ آپ void کے علاوہ کچھ لکھیں گے یعنی جس فنکشن کی ریٹرن ٹائپ void کے علاوہ کچھ ہوگی وہ ضرور کوئی نہ کوئی ویلیو ریٹرن کرتے ہیں۔ مثلاً

```
int ab( )
    {
    return a×b×c;
    }
```

یہ فنکشن a×b×c جو کہ ایک int ٹائپ کی ویلیو ہوگی ریٹرن کرتا ہے۔ آپ کو کوئی بھی ویلیو ریٹرن کرنے کے لئے کی ورڈ return لکھنا ہوگا۔ آیئے اس کی ایک مثال دیکھتے ہیں۔

مثال نمبر 2.18 فنکشن جو ویلیو ریٹرن کرتا ہے

```
//rest of Program
double volume( );
    {
    clrscr( );
    char choice;
```

```
double ans;
do
  {
  ans=volume( );
  cout <<"volume=" <<ans <<endl;
  cout <<"Do you want to continue:";
  cin >>choice;
  }
while(choice=='y'||choice=='y');
getch( );
}
double volume( )
{
double width, height, depth, result;
cout <<"Enter width, height, depth to find volume";
cin >>width >>height >>depth;
result=width*height*depth;
return width*height*depth;
}
```

آپ نے یہاں پر فنکشن کی ریٹرن ٹائپ ڈبل ظاہر کی ہے۔ اب آپ کو ڈبل ٹائپ ہی کی ویلیو ریٹرن کرنا ہوگی اگر آپ char یا Boolean ویلیو ریٹرن کریں گے تو یہ ناممکن ہوگا۔

اس کے علاوہ جو ویری ایبل فنکشن کی ریٹرن کی ہوئی ویلیو کو موصول کر رہا ہے یا جس میں آپ اسے سٹور کر رہے ہیں اسی کی ڈیٹا ٹائپ اور فنکشن کی ریٹرن ٹائپ ایک ہونی چاہئے یا پھر وہ ایک دوسرے سے compatiable ہوں ورنہ رزلٹ یعنی ریٹرن کی ہوئی ویلیو میں فرق بھی آ سکتا ہے۔ مثلاً آپ float ویلیو ریٹرن کرتے ہیں اور اسے سٹور int ٹائپ کے ویری ایبل میں کر دیتے ہیں تو اس سے ویلیو (ہوسکتا ہے اعشاریہ کے بعد والی ویلیو) میں کمی بیشی بھی ہو سکتی ہے۔ آپ جب اس پروگرام کو ایگزیکیوٹ کریں گے تو اس کی یہ آؤٹ پٹ ہوگی۔

```
Enter width, height, depth to find volume: 21  36  27
volume = 1512
Do you want to continue : n
```

اس پروگرام میں ہم نے ایک فنکشن ( )volume بنایا ہے جس کی ریٹرن ٹائپ ڈبل ہے۔ اس میں یوزر سے تین ان پٹ لی ہیں پھر ان کو ضرب دے کر رزلٹ کے ویری ایبل میں موصول کردہ ویلیو سٹور کی ہے۔ بعد میں یہ رزلٹ ویلیو کو ریٹرن کیا گیا ہے۔ جسے ( )main میں ڈبل ٹائپ کے ویری ایبل ans میں سٹور کروایا گیا ہے۔ یہاں پر ہم نے یہ ویلیو do-while لوپ میں موصول کی ہے۔ یہ لوپ یوزر سے پوچھتا ہے کہ کیا وہ دوبارہ یہی عمل دہرانا چاہتا ہے یا نہیں؟

پیرامیٹر لسٹ:

پیرامیٹرز کسی بھی فنکشن کے نام کے آگے بریکٹس ( ) میں لکھے جاتے ہیں۔ان بریکٹس میں ویری ایبلز کے نام اور ڈیٹا ٹپ لکھتے ہیں اور بعد میں انہیں ویلیو دی جاتی ہے۔اس کے دو الگ الگ نام ہیں اور یہ کافی کنفیوزنگ پوائنٹ ہے۔بعض لوگ انہیں ایک ٹیکنالوجی تصور کرتے ہیں لیکن یہ غلط ہے۔ ان کو پیرامیٹرز اور آرگومنٹس کہتے ہیں۔آیئے ان میں بنیادی فرق دیکھتے ہیں۔

- پیرامیٹر فنکشن کی پروٹو ٹائپ کے وقت لکھے جاتے ہیں کہ کتنے پیرامیٹرز پاس کئے جائیں گے اور ان کی ٹائپ کیا ہوگی۔البتہ اس کے ویری ایبلز کا نام یہاں نہیں لکھا جاتا۔ویری ایبلز کا کام اور ڈیٹا ٹائپ فنکشن کی ڈیفی نیشن کے وقت لکھی جاتی ہے۔یہ فنکشن کی باڈی میں استعمال ہوتے ہیں اور فنکشنز کو پاس کی جانے والی ویلیو کا ریفرنس نہیں ہوتے ہیں۔
- جبکہ آرگومنٹ ویلیو کو کہتے ہیں یعنی جو فنکشن کو پاس کی جاتی ہے۔یہ ویلیو فنکشن کی ایگزیکیوشن کے ٹائم پاس کی جاتی ہے یعنی جب فنکشن کال کیا جاتا ہے تو یہ ویلیو پاس کی جاتی ہے۔

C++ میں یوزر ڈیفائن فنکشنز کو آپ پیرامیٹرز کیسے پاس کر سکتے ہیں؟ آیئے اس کی ایک مثال دیکھتے ہیں۔

مثال نمبر 2.19 پیرامیٹر لسٹ کا استعمال

```
//Rest of Program or header files
void stdiam(double,double,double);
voud main( )
    {
    clrscr( );
    double width, depth, height;
    cout <<"Main Program" <<endl;
    cout <<"Function calling with argument passing" <<endl;
    stdiam(15, 7, 9);
    getch( );
    }
void stdiam (double w, double d, double h)
    {
    width = w;
    height = h;
    depth = d;
    cout <<w <<"*" <<d <<"*" <<h <<"=" <<(width*height*depth);
    }
```

جب آپ اس پروگرام کو ایگزیکیوٹ کریں گے تو اس کا رزلٹ یہ ہوگا۔

```
Main Program
Function calling with argument passing
```

15 * 7 * 9 = 945

اس پروگرام میں ہم نے تین گلوبل ویری ایبلز width, height, depth ڈبل ٹائپ کے ڈیکلیر کئے ہیں اور ایک فنکشن stdiam( ) بنایا ہے جس میں ڈبل ٹائپ کے تین پیرامیٹرز پاس کئے ہیں۔ آیئے ایک اور مثال دیکھتے ہیں۔

مثال نمبر 2.20 ریٹرن ٹائپ اور پیرامیٹر لسٹ

```
//header files
int volume( );
void values(int,int);
main( )
    {
    clrscr( );
    int     number1;
    int     number2;
    int     answer;
    char        ch;
    do
        {
        cout <<"Main method" <<endl;
        cout <<"Function calling" <<end;
        values (10, 31);
        answer=volume( );
        cout <<"Result is:" <<answer;
        cout <<endl; <<"Do you want to countinue?";
        cin >>ch;
        }
            while(ch=='y'||ch=='y');
     cout <<"End of Program";
     getch( );
     }
 void values(int r, int h)
     {
     number1 = r;
     number2 = h;
     }
```

```
int volume( )
    {
    return number1*number1*number2;
    }
```

اس پروگرام کی آؤٹ پٹ یہ ہوگی۔

```
Main method
Function calling
Result is = 3100
Do you want to continue?
n
End of Program
```

اس پروگرام میں ہم نے دو فنکشنز بنائے ہیں ایک جو ویلیو ریٹرن کرتا ہے int value( ) اور دوسرا جس میں سے پیرامیٹر پاس کے لئے ہیں۔ values(int,int) اس میں ہم نے دو int ٹائپ کے پیرامیٹرز پاس کئے ہیں۔ اس کے بعد ہم نے main( ) پروگرام میں دونوں فنکشنز do–while کے لوپ میں call کئے ہیں۔ do کی لوپ ہم نے یوزر کو ایک اضافی سہولت دینے کے لئے استعمال کیا ہے۔ اس کی مدد سے ہم یوزر سے یہ آپشن پوچھ رہے ہیں کہ کیا آپ یہ پروگرام دوبارہ ایگزیکیوٹ کرنا چاہتے ہیں یا نہیں۔ اگر یوزر y یا Y لکھے گا تو آؤٹ پٹ سکرین پر دوبارہ رزلٹ آجائے گا ورنہ پروگرام بند ہو جائے گا۔

اس کے علاوہ آپ ایسا فنکشن بھی لکھ سکتے ہیں جو کوئی ویلیو ریٹرن بھی کرتا ہو اور آرگومینٹ بھی پاس کرتا ہو۔ کیسے؟ آیئے اس کی ایک مثال دیکھتے ہیں۔

مثال نمبر 2.21 فنکشن میں آرگومینٹ پاسنگ اینڈ ریٹرن ٹائپ کا استعمال

```
//header files
int factor(int);
void main(void)
    {
    int no, result;
    cout <<"Enter any Positive number to find factorial:";
    cin >>no;
    result = factor(no);          //function calling
    cout <<"Factorial=" <<result;
    getch( );
    }
int factor(int fxo)
    {
```

```
int a, temp=1;
for (a=fxo; a>0; a--)
temp = a*temp;
return temp;
}
```

اس پروگرام میں ہم نے ایک فنکشن factor( ) کے نام سے بنایا ہے جو ایک ویلیو int ٹائپ کی ریٹرن کر رہا ہے اور int ٹائپ کی ویلیو پاس کر رہا ہے۔ اس پروگرام میں ہم ایک نمبر یوزر سے بطور ان پٹ لے رہے ہیں اس کا ہم فیکٹریوں معلوم کر رہے ہیں۔ اس پروگرام کی آؤٹ پٹ یہ ہوگی فرض کریں آپ نمبر 4 لکھتے ہیں۔

```
Enter any Positive number to find factorial : 4
Factorial = 24
```

اب تک آپ نے ایک ہی ٹائپ کی ویلیو ریٹرن اور پاس کی ہے۔ آپ دو الگ الگ ٹائپ کی ویلیوز بھی ریٹرن کر سکتے ہیں۔

مثال نمبر 2.22

```
#header files
char grade(int);
void main(void)
    {
    clrscr( );
    int marks=0;
    cout <<"Enter Marks";
    cin >>Marks;
    char obt-grade;
    obt-grade = grade(marks);
    cout <<"Your grade is:" <<obt-grade;
    getch( );
    }       //end of main
//Function defination
char grade(int l-marks)
    {
    switch (l-marks)
        {
        case 90:
        return 'A';
        break;
```

```
        case 80:
        return 'B';
        break;
        case 70:
        return 'C';
        break;
        default;
        return 'F';
    }  //end of switch
        }   //end of function
```

پرائم نمبر معلوم کرنا:

آیئے اب ایک ایسا پروگرام لکھتے ہیں جو پرائم نمبر معلوم کرتا ہوگا۔ پرائم نمبر ایسا نمبر ہوتا ہے جو صرف خود پر تقسیم ہوسکتا ہے۔ مثلاً 1,3,5,7 وغیرہ۔

مثال نمبر 2.23 پرائم نمبر معلوم کرنا

```
#include <math.h>
#include <iostream.h>
#include <canio.h>
int primeno(int);
void main( )
    {
    for(int i=1; i<40; i++)    {
        if(Primeno(i))
        cout <<i <<"\t";
        }   //end of for loop
    }   //end of main
int Primeno(int i)
    {
    float temp;
    temp = sqrt(i);
    if (i<2) return 0;
    if (i==2) return 1;
    if (1%2==0) return 0;
    for(int a=3; a<=temp; a+=2)
```

```
    if (i%a==0) return 0;
    return 1;
    }
```

اس پروگرام میں ہم نے فنکشن میں آنے والے نمبر i کو a سے تقسیم کیا ہے۔ موڈ (%) آپریٹر کی مدد سے ہم نے یہ if کنڈیشن (i%a==0) لگائی ہے۔ اگر ایسا ہوا تو یہ نمبر i پرائم نمبر نہیں ہوگا۔ یہاں سے اس کی ویلیو 0 یعنی false ہو جائے گی۔ اور اگر ایسا نہ ہوا تو پھر یہ 1 یعنی true ریٹرن کرے گا کہ یہ نمبر پرائم ہے۔ اس پروگرام کی آؤٹ پٹ یہ ہوگی۔

2 3 5 7 11 13 17 19 23 31 37

اب ہم ایک ایسا پروگرام لکھیں گے جو لیپ ایئر (Leap year) کے بارے میں معلومات دے گا۔ لیپ ایئر ایسے سال کو کہتے ہیں جس میں عام سالوں سے ایک دن زیادہ ہوتا ہے۔ بعض لوگ یہ کہتے ہیں کہ جو سال چار (4) پر تقسیم ہو جائے وہ لیپ ایئر ہوتا ہے لیکن 1800, 1900 یا 1700 لیپ ایئر نہیں تھے لیکن اس میں بھی ایک مسئلہ ہے کہ 2000 لیپ سال تھا۔ تو یوں centennial سال بھی لیپ ایئر ہو سکتے ہیں۔ اب اس میں یہ ہے کہ ایسا سال جو چار پر تقسیم ہو لیپ ایئر ہوگا۔ لیکن centennial سالوں کے لئے یہ اصول نہیں ہے۔ ایسا centennial سال جو 400 پر تقسیم ہو جائے وہ لیپ سال ہوگا۔ مثلاً 2000 وغیرہ۔ تو آیئے اس کے لئے ایک پروگرام لکھتے ہیں۔

مثال نمبر 2.24 لیپ ایئر معلوم کرنا

```
//header files
int isleap(int);
void main( )
    {
    clrscr( );
    int temp;
    char choice;
    do
        {
        cout <<"Enter year:";
        cin >>temp;
        if(isleap (temp))
        cout <<temp <<"is not a leap year \n";
        else
        cout <<temp <<"is a leap year \n";
    }   while (temp>1);
    getch( );
    }
int isleap(int a)
```

```
{
return a%4==0&&a%100!=0||a%400==0;
}
```

پیرامیٹر بائی ریفرنس پاس کرنا:

اب تک آپ نے فنکشن میں تمام پیرامیٹرز ویلیو کے ذریعے پاس کئے ہیں یعنی فنکشنز میں ابھی تک ویلیوز پاس کی ہیں۔ اس کا یہ مطلب ہے کہ پہلے ویلیو پر آپریشن پرفارم ہوگا اور پھر فنکشن ایگزیکیوٹ ہونے سے پہلے ویلیو کسی متعلقہ پیرامیٹر کو آسائن کی جائے گی۔ مثلاً value(a) فنکشن کال کیا جاتا ہے۔ اب فرض کریں کہ a=3 یعنی a کی ویلیو 3 ہے تو فنکشن کے ایگزیکیوٹ ہونے سے پہلے یہ ویلیو کسی لوکل ویری ایبل کو آسائن کرنا ہوگی۔ اس کا مطلب یہ ہے کہ ویری ایبل پر فنکشن کا کوئی اثر نہیں ہے۔ اس لئے ویری ایبل a Read-only پیرامیٹر ہے۔

بعض صورتِ احوال ایسی بھی ہوتی ہیں کہ آپ کو فنکشن کے پیرامیٹر کی ویلیو تبدیل کرنے کی ضرورت پیش آسکتی ہے تو ایسا آپ پیرامیٹر بائی ریفرنس سے کر سکتے ہیں۔ بائی ریفرنس پیرامیٹر بھیجنے کے لئے فنکشن کی بریکٹس میں ڈیٹا کے ساتھ اینڈ علامت (&) (جسے ampersand بھی کہتے ہیں) لکھی جاتی ہے۔ اس کی وجہ سے لوکل ویری ایبل اصل پیرامیٹر کا ایک ریفرنس بن جاتا ہے۔ اس سے اصل پیرامیٹر Read-only کی بجائے Read-write بن جاتا ہے۔ اس کا فائدہ یہ ہے کہ فنکشن لوکل ویری ایبل میں تبدیلی کا اثر اصل پیرامیٹر پر بھی ہوگا جو اسے پاس کرتا ہے۔ آیئے اس کی ایک مثال دیکھتے ہیں۔

مثال نمبر 2.25 پیرامیٹر بائی ریفرنس

```
//header files
void result(int&,int&);
void main( )
    {
    clrscr( );
    int temp1=71, temp2=31;
    int temp3=80, temp2=64;
    result(temp1,temp2);
    result(temp3,temp4);
    cout <<"First time call to result:" <<temp1 <<temp2 <<endl;
    cout <<"Second time call to result:" <<temp3 <<temp4 <<endl;
    getch( );
    }
void(int& n1, int& n2)
    {
    if(n1>n2)
        {
        int temp;
```

```
temp=n1;
n1=n2;
n2=temp;
}
```

جب آپ اس پروگرام کو ایگزیکیوٹ کریں گے تو یہ رزلٹ ڈسپلے ہوگا۔

```
First time call to result:  31  71
Second time call to result: 64  80
```

اس پروگرام میں ہم نے ایک فنکشن کو دو ٹائمز کال کیا ہے۔

اوپر آپ نے ایک پروگرام لکھا ہے جس میں پیرامیٹر بائی ریفرنس پاس کئے گئے ہیں۔ آیئے اب ایک ایسا پروگرام لکھتے ہیں جس میں ایک پیرامیٹر بائی ریفرنس کا ہوگا اور دوسرا بائی ویلیو ہوگا۔ اس سے آپ کو ان دونوں میں واضح فرق بھی معلوم ہو جائے گا۔

مثال نمبر 2.26 بائی ریفرنس اینڈ بائی ویلیو پیرامیٹرز

```
void differ(int x,int& y);
  {
  x = 72;
  y = 42;
  }
void main( )
  {
  clrscr( );
  int i=10, j=39;
  cout <<"i=" <<i <<",j=" <<j <<endl;
  differ(i,j);
  cout <<After function call" <<endl;
  cout <<"i=" <<i <<",j=" <<j <<endl;
  getch( );
  }
```

اس پروگرام کی آؤٹ پٹ یہ ہے۔

```
i = 10, j = 39
After function call
i = 10, j = 42
```

اس پروگرام میں (differ(i,j کال کیا گیا ہے اس میں i بائی ویلیو x کو پاس کیا جا رہا ہے اور j بائی ریفرنس y کو پاس کیا ہے۔ x ایک لوکل ویری ایبل ہے جس کو i کی ویلیو 10 آسائن کی گئی ہے جبکہ y ویری ایبل j کا متبادل (alias) ہے جس کی ویلیو 42 ہے۔ فنکشن 72 ویلیو i کو آسائن کرتا ہے لیکن

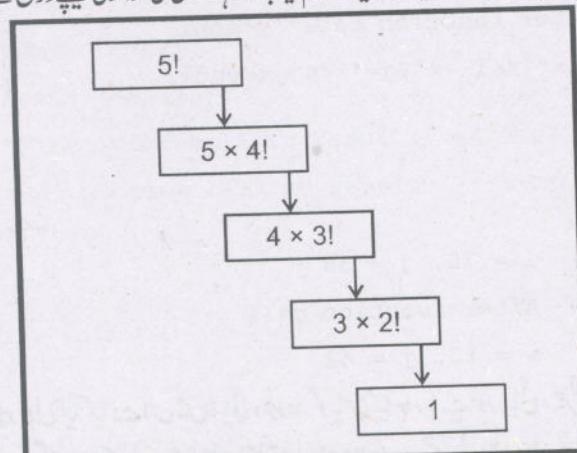
اس کا اثر کوئی اثر نہیں ہے لیکن جب y ویری ایبل 99 ویلیو j کو آسائن کرتا ہے تو یہ j پر اثر انداز ہوتا ہے۔ اس لئے جب آپ پروگرام ایگزیکیوٹ کرتے ہیں تو i کی اصل ویلیو 10 پرنٹ ہوتی ہے جبکہ j کی ویلیو فنکشن میں موجود y کی ویلیو 42 پرنٹ ہوتی ہے۔

پیرامیٹر بائی ریفرنس پاس کرنے کے دو اہم فوائد ہیں۔ پہلا یہ کہ اگر آپ پیرامیٹر کی ویلیو تبدیل کرنا چاہتے ہیں تو پیرامیٹر بائی ریفرنس پاس کر سکتے ہیں۔ اس کے علاوہ فنکشن کے پیرامیٹر میموری میں بہت زیادہ جگہ گھیرتے ہیں۔ تو اس کے لئے بہترین طریقہ پیرامیٹر کو بائی ریفرنس پاس کرنے کا ہے۔ اس کے علاوہ پیرامیٹر بائی ریفرنس پاس کرنے کا ایک فائدہ یہ ہے کہ اگر آپ پیرامیٹر کی ویلیو تبدیل نہیں کرنا چاہتے تو اسے کانسٹنٹ ریفرنس پاس کر سکتے ہیں۔ یہ پیرامیٹر وہی کام کرے گا جو پیرامیٹر بائی ریفرنس کرے گا۔ فرق صرف اتنا ہے کہ اس میں آپ پیرامیٹر کی ویلیو تبدیل نہیں کر سکتے۔ اس کو لکھنے کا طریقہ یہ ہے۔

```
int factor(int x, int y, const int& tem0);
```

یعنی جو بھی پیرامیٹر آپ کانسٹنٹ بائی ریفرنس پاس کرنا چاہتے ہیں ان کے ساتھ const کی ورڈ لکھنا ہوگا۔ اور بعد میں آپ فنکشن باڈی میں اس ویلیو کو تبدیل نہیں کر سکتے۔

ریکریژن: (Recursion)

اب تک آپ نے جتنے فنکشنز پڑھے ہیں آپ ان کو ایک دفعہ یا دو دفعہ main( ) سے کال کرتے رہے ہیں۔ اس کے علاوہ C++ میں ایک اور ٹیکنالوجی استعمال ہوئی ہے جسے ریکرژو فنکشن کہتے ہیں۔ ایسا فنکشن جو خود کو ڈائریکٹ یا ان ڈائریکٹ کال کرے خواہ وہ خود کو کال کر رہا ہو ریکوژو فنکشن کہلاتا ہے۔ ایسا فنکشن بنیادی پرابلمز کو حل کرنے کے لئے لکھا جاتا ہے اور اگر پرابلم مشکل ہو تو یہ اس کو چھوٹے چھوٹے ٹکڑوں میں حل کرتا ہے۔ مثلاً اگر پرابلم بنیادی ہوگی تو یہ فنکشن رزلٹ ویلیو ریٹرن کر دے گا اور اگر پرابلم مشکل ہوگی تو یہ اسے دو حصوں میں تقسیم کرے گا۔ ایک ایسا حصہ جسے فنکشن جانتا ہے کہ کیا کرنا ہے اور کیسے یہ حل ہوگا۔ اور دوسرا جس کے بارے میں فنکشن کو معلوم نہیں ہے کہ اسے کیسے حل کرنا ہے۔ ریکریژن کی مثال اس سے سمجھ سکتے ہیں۔

```
n! = n(n-1)!
```

یہ کسی بھی نمبر کا فیکٹوریل معلوم کرنے کا فارمولہ ہے اور اس کی مدد سے آپ کسی بھی نمبر کا فیکٹوریل معلوم کر سکتے ہیں۔ یہ اس طرح کام کرتا ہے۔

```
5!
5×(4!)
```

یہ پروگرام کس طرح کام کرتا ہے یا ایک نمبر کا فیکٹوریل نمبر کیسے معلوم کیا جاتا ہے۔ اس کی حرآر کی نیچے درج ہے۔

یہ اس طرح اس وقت تک کال ہوتا رہتا ہے جب تک کہ آخری ویلیو 1 نہیں ہو جاتی۔ آیئے اس کی ایک مثال دیکھتے ہیں۔

مثال نمبر 2.27 ریکرژ فنکشن

```
#include<conio.h>
#include<iostream.h>
#include<iomanip.h>
unsigned long factor(unsigned long);
void main(void)
    {
    for(int i=1; i<=10; i++)
    cout <<i <<"!=" <<setw(3) <<factor(i) <<endl;
    }
unsigned long factor(unsigned long a)
    {
    if(a<=0)
    return1;
    else
    return a*factor(a-1);
    }
```

اس پروگرام کی آؤٹ پٹ کچھ یوں ہوگی۔

```
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5042
8! = 40320
9! = 362880
10! = 362880
```

اس طرح آپ جس نمبر کا چاہیں فیکٹوریل معلوم کر سکتے ہیں۔اس کے علاوہ آپ اس فنکشن کی مدد سے Fibonacci سیریز بھی معلوم کر سکتے ہیں۔ Fibonacci سیریز یہ ہوتی ہے۔

0, 1, 1, 2, 3, 4, 5, 8, 13, 21, ...

یہ سیریز صرف (0) سے شروع ہوتی ہے اور اس کے بعد ہر نیا نمبر پچھلے دو نمبرز کو جمع کر کے نکالا جاتا ہے۔اس کا فارمولا یہ ہے۔

```
fabonacci(n)=fibnacci(n-1)+fabonacci(n-2)
```

آیئے اب اس کی ایک مثال لکھتے ہیں۔ جس میں یوزر کوئی بھی نمبر تحریر کرے گا اور ہم اس کی fibonacci سیریز معلوم کریں گے۔

مثال نمبر 2.28 Fibonacci سیریز معلوم کرنا

```
//header files
long fiboseries(long);
void main( )
    {
    clrscr( );
    long answer, temp;
    cout <<"Enter a number";
    cin >>temp
    answer=fiboseries(temp);
    cout <<"fibonacci("<<temp<<")="
        result <<endl;
    getch( );
    }
long fiboseries(long a)
    {
    if(a==0¦¦a==1)
    return a;
    else
    return fiboseries(n-1)+fiboseries(n-2);
    }
```

اس پروگرام کو ایزیکیوٹ کریں اور رزلٹ پر غور کریں۔ فرض کریں کہ یزور 10 نمبر لکھتا ہے۔

```
Enter a number: 10
fibonacci(10) = 55
```

اس میں یہ ہے کہ آپ کے پاس پہلے دس نمبر کے بعد یہ نمبر 55 آئے گا۔ اس طرح آپ کسی بھی نمبر کا رزلٹ معلوم کر سکتے ہیں۔

فنکشن اوور لوڈنگ:

C++ میں آپ ایک ہی کام کے کئی فنکشن بنا سکتے ہیں لیکن ان کی پیرامیٹر لسٹ مختلف ہونی چاہئے۔ یعنی فنکشنز کے نام ایک ہو سکتے ہیں لیکن پیرامیٹر لسٹ مختلف ہوگی تو C++ اسے الگ الگ فنکشن تصور کرے گی۔ اس کو فنکشن اوور لوڈنگ کہتے ہیں۔ جب ایسے فنکشن کال کئے جاتے ہیں تو C++ کمپائلر اس کے پیرامیٹرز کی ٹائپ، ویلیو اور تعداد سے ان میں تمیز کرتا ہے کہ کون سا فنکشن کال کیا گیا ہے۔ آیئے اس کے لئے ایک پروگرام کی مدد حاصل کرتے ہیں۔

مثال نمبر 2.29 اوورلوڈ فنکشنز

```
//header files
int result(int,int);
int result(int,int,int);
void main( )
    {
    clrscr( );
    cout <<"Sum of two numbers=" <<result (15,7);
    cout <<endl;
    cout <<"Sum of three numbers=" <<result (7,13,8);
    getch( );
    }
int result(int a, int b, int c)
    {
    return a+b+c;
    }
int result(int a, int b)
    {
    result a+b;
    }
```

اس پروگرام میں دو فنکشنز ایک ہی کام result سے بنائے گئے ہیں لیکن ان کے پیرامیٹر کی تعداد مختلف ہے۔ اس لئے C++ کمپائلر انہیں الگ الگ فنکشن تصور کرے گا۔ ایک بات اور نوٹ کریں کہ ریٹرن ٹائپ کا اس پر کوئی اثر نہیں ہوتا۔ اب اگر آپ تین پیرامیٹرز والا فنکشن کال کرنا چاہتے ہی تو فنکشن کال کے وقت تین ویلیو تحریر کریں۔ اس کے علاوہ فنکشن کے پیرامیٹر کی تعداد بھی ایک جیسی ہو سکتی ہے لیکن اس کے لئے یہ ضروری ہے کہ ان کی ڈیٹا ٹائپ پہلے فنکشن سے مختلف ہو۔ مثلاً

```
int result (int,int);
double result(int, char);
```

اوپر والے پروگرام میں ہم نے دو پیرامیٹرز والا فنکشن کال کیا ہے اور بعد میں تین پیرامیٹرز والا فنکشن کال کیا ہے۔ یوں اوپر والے فنکشن کی آؤٹ پٹ یہ ہوگی۔

```
Sum of two numbers = 22
Sum of three numbers = 28
```

ڈیفالٹ آرگومنٹس:

C++ میں آپ کئی آرگومینٹس دے سکتے ہیں اس کے لئے آپ کو ڈیفالٹ ویلیو تحریر کرنا ہوں گی۔ آپ سوچ رہے ہوں گے کہ یہ کیا ہے اور کس

طرح ہوگا؟ تو آیئے اس کی ایک مثال دیکھتے ہیں۔

مثال نمبر 2.30

```
//header files
float result(float,float=0,float=0)
void main( )
    {
    float temp=1.02;
    cout <<"result(temp,3)=" <<result(temp,3) <<endl;
    cout <<"result(temp,3,5)=" <<result(temp,3,5) <<endl;
    cout <<"result temp=" <<result(temp) <<endl;
    getch( );
    }
float result(float a, float b, float c)
    {
    return a+(b+(c*a*b)*c);
    }
```

اس پروگرام کی آؤٹ پٹ یہ ہوگی۔

```
result(temp,3)=7.1412
result(temp,3,5)=12.241199
result(temp)=1.02
```

اس پروگرام میں ہم نے فنکشن کے دو پیرامیٹرز کو ڈیفالٹ کیا ہے جس کی وجہ سے ہم ان دو پیرامیٹرز کی ویلیوز اگر درج نہ کرنا چاہیں تو تب بھی ممکن ہے اور آپ نے دیکھا کہ ہم نے اوپر مثال میں پہلے دو ویلیوز دی ہیں پھر تین اور آخر پر صرف ایک ویلیو تحریر کی ہے۔

مشق

سوال نمبر 1: ایک ایسا پروگرام لکھئے جو اس طرح کی آؤٹ پٹ ڈسپلے کرواتا ہو؟

```
(a)  *                    (b) * * * * * *
     * *                        * * * * *
     * * *                        * * * *
     * * * *                        * * *
     * * * * *                        * *
     * * * * * *                        *
```

سوال نمبر 2: ایک ایسا پروگرام لکھیں جو ایک سادہ کیلکولیٹر کا کام کرے یعنی دو نمریک ویلیو یوزر سے لے کر اور آپریٹر بھی یوزر سے مانگے کہ وہ ان دو ویلیوز پر کون سا آپریٹر پرفارم کرنا چاہتا ہے۔ یہ سوئچ سٹیٹ مینٹ کی مدد سے حل کریں۔

سوال نمبر 3: ایک ایسا پروگرام تحریر کریں جس میں یوزر سینٹی گریڈ درجہ حرارت کو فارن ہائیٹ یا فارن ہائیٹ کو سینٹی گریڈ میں تبدیل کر سکے۔

سوال نمبر 4: اس پروگرام کی آؤٹ پٹ کیا ہوگی؟

```
for(int i=1; i<5; i++)
  {
  for(int j=0; j<i; j++)      {
     for(k=0; k<j; k++)
     cout <<"+";
     cout <<end;l
     }
  cout <<endl;
  }
```

سوال نمبر 5: اس مساوات کو فنکشن کی مدد سے حل کریں۔

```
(n,k)=(n-1)(n-2)........(n-k+2)(n-k+1)
```

اس کی آؤٹ پٹ یوں ہوگی۔

```
0  0
0  1  0
0  1  1  0
```

```
0  1  2  2  0
0  1  3  6  6  0
0  1  4 12 24 24 0
```

سوال نمبر 6: ایک ایسا پروگرام لکھیں جو ایک نمبر n کی پاور معلوم کرے۔ یہ نمبر n یوزر تحریر کرے اور اس کی پاور p میں آپشن ہو کہ وہ یوزر خود واضح کرنا چاہتا ہے یا نہیں۔ اگر یوزر وہ پاور خود تحریر نہ کرے تو پھر یہ پاور p کی ویلیو 3 ہو۔

سوال نمبر 7: آپ نے اس مشہور فارمولہ کے بارے میں پڑھا ہوگا۔ اس کو حل کریں۔

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

یہ quadratic مساوات کا فارمولہ ہے یہ مساوات یوں ہوتی ہے۔ $x^2+bx+c=0$ یوں یوزر تین نمبر تحریر کرے گا اور اس فارمولہ کی مدد سے حل کئے جانے چاہئیں۔

سوال نمبر 8: یوزر سے ایک ان پٹ لیں اور معلوم کریں کہ کیا وہ پرائم نمبر ہے یا نہیں اور کیا یوزر مزید اس پروگرام کو جاری رکھنا چاہتا ہے یا نہیں؟

سوال نمبر 9: مختصر جواب دیں۔

(i) while اور do-while میں کیا فرق ہے؟

(ii) if, switch اور else-if کس لئے استعمال ہوتے ہیں؟

(iii) ریکرژ فنکشن کیا ہوتا ہے؟

❶: جواب

Ans (a)

```
#include<conio.h>
#include<iostream.h>
void main( )
    {
    clrscr( );
    for(int i=0; i<6; i++)
        {
        for(int j=0; j<6; j++)
        cout <<"*";
        cout <<endl;
        }
    getch( );
    }
```

Ans (b)

```
#include<conio.h>
#include<iostream.h>
void main( )
    {
    clrscr( );
    for(int i=6; i>0; i--)
        {
        for(int j=6; j>0; j--)
        cout <<"*";
        cout <<endl;
        }
    getch( );
    }
```

❷: جواب

```
#include<conio.h>
#include<iostream.h>
#include<iomanip.h>
void main( )
    {
    clrscr( );
    int temp1,temp2;
    double result;
    char op;
    cout <<"Enter 1st number:";
    cin >>temp1;
    cout <<"Enter 2nd number:";
    cin >>temp2;
    cout <<"Enter operator(i.e +):";
    cin >>op;
        {
        case '+':
        result = temp1+temp2;
        cout <<"Answer=" <<result;
        break;
        case '-':
        result = temp1-temp2;
        cout <<"Answer=" <<result;
        break;
        case '*':
        result = temp1*temp2;
        cout <<"Answer=" <<result;
        break;
        case '/':
        result = (float)temp1/temp2;
        cout <<"Answer=" <<setprecision(2) <<result;
        break;
```

```
case '%':
result = temp1%temp2;
cout <<"Answer=" <<result;
break;
default:
cout <<"\n Sorry unknown operator:";
break;
}
getch( );
}
```

: جواب

```
#include<conio.h>
#include<iostream.h>
#include<iomanip.h>
void main( )
{
clrscr( );
double temp, result;
int option;
cout <<"Press 1 to convert celsius to fahrenheits:\n"
     <<"Press 2 to convert fahrenheits to celsius:";
cin >>option;
switch(option)
{
case 1:
cout <<"Enter temperature in celsius:";
cin >>temp;
result = (9.5/5.0*temp)+32;
cout <<"In fahrenheits temperature=" <<result;
break;
case 1:
cout <<"Enter temperature in fahrenheits:";
cin >>temp;
```

```
        result = (temp-32)*5/9;
        cout <<"In celsius temperature=" <<result;
        break;
        default;
        cout <<"Sorry wrong choice:";
        break;
        }
    getch( );
    }
```

4: اس پروگرام کی آؤٹ پٹ یوں ہوگی۔

```
    *

    *
    * *

    *
    * *
    * * *
```

5: جواب

```
#include<conio.h>
#include<iostream.h>
#include<iomanip.h>
int factor (int a)
{
    int f=1;
    if (a < 0)
    return 0;
    while (a > 1)
    f *=a--;
    return f;
    }
int permutation(int a, int b)
    {
```

```
if (a < 0 || b < 0 || b > a)
return 0;
else
return factor(a)/factor(a-b);
}
void main(void)
{
clrscr( );
int temp;
cout <<"how many permutation(s) you want:\n"
cin >>temp;
for(int i=-1; i (temp; i++)
{
for(int j=-1; j<=i+1; j++)
cout <<" " <<permutation(i,j);
cout <<endl;
}
getch( );
}
```

6: جواب

```
#inculde<conio.h>
#inculde<iostream.h>
int prime(int);
void main( )
{
clrscr( );
int temp, power, answer, choice;
cout <<"Enter a Number:";
cin >>temp;
cout <<"Press 1 to enter Power \n"
<<"otherwise press any key:";
cin >>power;
answer=pow(temp,power);
```

```
 cout <<"With power("<<temp <<"," <<power <<")\n Answer=" <<answer;
 }
else
 {
 cout <<"\n With default power 3:";
 answer=pow(temp,3);
 cout <<"\n power("<<temp <<",3) \n Answer=" <<answer;
 }
 getch( )
 }
```

❼: جواب

```
#inculde<conio.h>
#inculde<iostream.h>
#inculde<math.h>
void main(void)
    {
    clrscrI( );
    cout <<"Solving ax^2+bx+c=0" <<end;
    int a, b, c;
    double temp1, temp2, temp3;
    in:
    cout <<"Enter coefficient a:";
    cin >>a;
    cout <<"Enter coefficient b:";
    cin >>b;
    cout <<"Enter coefficient c:";
    cin >>c;
    if(a==0)
        {
        cout <<"This is not Quadratic Equation: Enter again";
        goto in;
        }
    cout <<"the Equation is: "<<a <<"x^2+" <<b <<"x+" <<c <<"=0\n";
```

```
temp1 = b*b-4*a*c;
    {
    cout <<"This Equation has no real solution:";
    getch( );
    exit(0);
    }
temp2 = (-b+sqrt(temp1))/(2*a);
temp3 = (-b-sqrt(temp1))/(2*a);
cout <<"The Two possible solution are=" <<temp2 <<"," <<temp3;
getch( )
}
```

8: پروگرام یوزر سے ان پٹ لے گا اور پھر بتائے گا کہ کیا یہ پرائم نمبر ہے یا نہیں۔

```
#inculde<conio.h>
#inculde<iostream.h>
int prime(int);
void main( )
    {
    clrscr( );
    int no;
    cout <<"Enter a number:";
    cin >>no;
    if(prime(no))
    cout <<no <<"is a prime number:";
    else
    cout <<no <<"is not a prime number:";
    getch( );
    }
int prime(int temp)
    {
    if(temp<2) return 0;
    if(temp==2) return 1;
    if(temp%2==0) return 0;
    for(int i=3; i<temp; i+=2)
```

```
if(temp%1==0) return 0;
return 1;
}
```

9: جواب

(i) while لوپ میں پہلے کنڈیشن چیک ہوتی ہے اور اس کے بعد لوپ باڈی ایگزیکیوٹ ہوتی ہے اور اگر کنڈیشن غلط ہوگی تو لوپ نہیں چلے گی۔ جبکہ do-while لوپ میں کم از کم ایک دفعہ لوپ ضرور ایگزیکیوٹ ہوتی ہے کیونکہ اس میں لوپ کنڈیشن بعد میں چیک ہوتی ہے۔

(ii) یہ تینوں کنٹرول سٹیٹ مینٹس ہیں۔ if سٹیٹ مینٹ میں آپ اپنی ایکسپریشن چیک کرتے ہیں کہ کیا وہ درست ہے یا نہیں۔ اگر کنڈیشن درست ہوگی تو اس سے متعلقہ سٹیٹ مینٹ ایگزیکیوٹ ہوگی ورنہ نہیں۔ else-if کنڈیشن if کنڈیشن کے فیچرز میں اضافہ کرتی ہے اس کی مدد سے آپ ایک سے زیادہ کنڈیشنز چیک کرتے ہیں یعنی پہلی کنڈیشن اگر غلط ہے تو دوسری چیک ہو۔

(iii) switch سٹیٹ مینٹ بھی کنٹرول سٹیٹ مینٹ ہے جس میں آپ ایک ایکسپریشن کی کئی ویلیوز کو چیک کر سکتے ہیں۔

(iv) ایسا فنکشن جو خود کو ڈائریکٹ یا ان ڈائریکٹ کال کرے۔ خواہ یہ فنکشن خود سے خود کو کال کر رہا ہو یا کسی دوسرے فنکشن سے خود کو کال کر رہا ہو ریکرژ فنکشن کہلاتا ہے۔

باب نمبر 3

اریز اینڈ سٹرنگ

ایک ارے ایسے اوبجیکٹس کی ترتیب کا ایک نام ہے جس کی ڈیٹا ٹائپ ایک ہونی چاہئے یہ اوبجیکٹس ارے کے اجزاء کہلاتے ہیں۔ ++C میں ارے کی بڑی اہمیت ہے اس میں آپ ایک ہی قسم کا ڈیٹا سٹور کر سکتے ہیں۔ ++C میں بھی اریز C کی طرح کام کرتی ہیں۔ اور یہ تقریباً ہر کمپیوٹر لینگویج میں ہوتی ہیں اس کے علاوہ آپ اس کتاب میں سٹرنگ کے بارے میں بھی پڑھیں گے۔ سٹرنگ کا یہ فائدہ ہے کہ اس میں آپ کئی الفاظ (یعنی ایک جملہ) محفوظ کر سکتے ہیں۔ اس کے علاوہ آپ سٹرنگز سے متعلق کئی اور فنکشن بھی پڑھیں گے۔ اس باب میں آپ مندرجہ ذیل فنکشنز پڑھیں گے۔

- اریز
- Linear سرچ
- بائنری سرچ
- اریز کو ترتیب دینا
- ملٹی پل سب سکرپٹس ارے
- سٹرنگز
- سٹرنگ لائبریری
- سٹرنگ جمع کرنا
- سٹرنگ کاپی کرنا
- مشق

اریز:

ایک ارے ایسے اوبجیکٹس کی ترتیب کا ایک نام ہے جس کی ڈیٹا ٹائپ ایک ہونی چاہئے یہ اوبجیکٹس کے اجزاء کہلاتے ہیں اور ہر اوبجیکٹ کا ایک مخصوص نمبر ہوتا ہے۔ اسے ارے کا انڈیکس کہتے ہیں اور اس کا دوسرا کام سب سکرپٹ بھی ہے۔ دوسرے الفاظ میں ارے ایسی ڈیٹا ٹائپ کو کہتے ہیں جس میں ایک ہی نام کے ویری ایبلز سٹور کئے جاتے ہیں اور یہ میموری میں ترتیب سے اکٹھی ایک ساتھ جگہ گھیرتی ہے۔ آپ ایک ارے یوں ڈیکلیر کر سکتے ہیں۔

```
int array[ ];
```

یہاں ہم نے ایک int ٹائپ کی ارے بنائی ہے جس کا نام ارے رکھا ہے۔ اب اس میں a[1] سے مراد وہ ویلیو ہوگی جو میموری میں ارے کی 1 پوزیشن پر محفوظ ہوگی۔ ارے میموری میں صفر (0) سے شروع ہوتی ہے۔ ارے میموری میں یوں جگہ گھیرتی ہے۔

int array[5]

10	12	8	7	13
0	1	2	3	4

اس سے آپ کو اندازہ ہو جائے گا کہ میموری میں ارے کس طرح جگہ گھیرتی ہے۔ ارے میں پہلی لوکیشن صفر ہوتی ہے یعنی آپ نے 6 ویلیوز کے لئے ارے ڈیکلیر کی ہے تو میموری میں ان کی لوکیشن 5-0 تک ہوگی۔ آخری ویلیو لوکیشن 5 میں محفوظ ہوگی۔

آجکل تقریباً تمام پروگرامرز ارے کا استعمال کرتے ہیں کیونکہ اس طرح آپ کا ٹائم کم ضائع ہوتا ہے اور اس کے علاوہ میموری بھی زیادہ ضائع نہیں ہوتی۔ مثلاً آپ اگر 100 طالب علموں کا ریکارڈ رکھنا چاہتے ہیں تو کیا ہر ایک الگ سے ویری ایبل ڈیکلیر کریں گے اور الگ الگ ان پٹ دیں گے۔ اس کے لئے آپ کو ارے کا استعمال کرنا ہوگا۔ آپ اپنے پروگرام میں ارے کو یوں شامل کر سکتے ہیں۔

مثال نمبر 3.1 int ارے کا استعمال

```
void main( )
    {
    int count[4], a=0;
    clrscr( );          //to clear screen
    count <<"Enter 4 integers";
    while(a<5) {
        cout <<"Enter digit no" <<a+1;
        cin >>count[a];
        a++;
        }
     a=0;
     do  {
        cout <<"\n" <<count[a];
        a++;
        }   while(a<5);
```

```
getch( );
}
```

اس پروگرام میں یوزر سے ان پٹ لی جارہی ہے اور یہ ان چار دفعہ (یعنی چار ان پٹس لی جارہی) ہیں مگر ایک اہم بات یہ ہے کہ <<cin صرف ایک دفعہ کروایا ہے اس لئے کہ یہ ان پٹ ارے کی مدد سے لوپ میں لی جارہی ہے اور بعد میں ان نمریک نمبرز کو کاؤنٹ کیا گیا ہے۔
اس طرح آپ ایک نام یعنی کریکٹر ارے بھی لے سکتے ہیں۔اس کا یہ فائدہ ہے کہ آپ اس میں ایک سٹرنگ بھی لے سکتے ہیں۔char ڈیٹا ٹائپ میں صرف کریکٹر لکھا جاسکتا ہے اور سٹرنگ کے لئے آپ کو ارے ڈیکلیر کرنا ہوگی اور سٹرنگ ان پٹ کے لئے جو میتھڈ استعمال ہوتا ہے اس کا ذکر ہم پہلے ہی کر چکے ہیں۔آیئے ایک سٹرنگ ارے دیکھتے ہیں۔

```
{
clrscr( );
char name[20];
cout <<"Enter Your Name";
gets(name);
getch( );
}
```

اس پروگرام میں ہم نے ایک کریکٹر ارے ڈیکلیر کی ہے جس کی لمبائی 20 ہے۔ہم نے ( )clrscr کا فنکشن استعمال کیا ہے۔ان پٹ سکرین صاف کرتا ہے اور ان پٹ کے لئے ( )gets استعمال کیا ہے۔آیئے ارے کا ایک اور پروگرام لکھتے ہیں۔

مثال نمبر 3.2 ارے ڈیکلیریشن اینڈ انیشلائیزیشن

```
void main( )
{
int no[10]={0,3,5,7,9,11,13,15,17,19};
cout <<"index number" <<setw(15) <<"value" <<endl;
for(int a=0; a<10; a++)
cout <<setw(9) <<a <<setw(15) <<no[a] <<endl;
getch( );
}
```

اس پروگرام میں ہم نے ( )setw کا فنکشن استعمال کیا ہے اس کے لئے آپ کو ہیڈر فائل شامل کرنا ہوگی جس کا نام یہ ہے۔

```
#include<iomanip.h>
```

( )setw کی مدد سے آپ دو آؤٹ پٹس کے درمیان فاصلہ اپنی مرضی سے سیٹ کر سکتے ہیں۔اس پروگرام کی آؤٹ پٹ یہ ہوگی۔

```
Index no        Value
0               0
1               3
2               5
```

3	7
4	9
5	11
6	13
7	15
8	17
9	19

آیئے ارے کی ایک اور مثال دیکھتے ہیں۔

```
void main( )
  {
  const int array1=7;
  int count[array1]={15,8,2,7,11,5,3};
  cout <<"array index" <<setw(12) <<"value"
  <<setw(12) <<"Magic Histogram" <<endl;
  for(int i=0; i<array1; i++)
  {
  cout <<setw(12) <<count[i] <<setw(12);
  for(int j=0; j<count[i]; j++)
  cout <<"*";
  cout <<endl;
  }
  getch( );
  }
```

مثال نمبر 3.3

اس پروگرام میں ایک کانسٹنٹ ویری ایبل array1 لیا ہے۔اس کی ڈیٹا ٹائپ int ہے اور اس کی ویلیو 7 ہے اور ایک int ٹائپ کی ارے بنائی ہے جس کی لمبائی array1 ہے اور اسے انی شلائیز کر دیا گیا۔اس کے بعد nested for لوپ استعمال کی ہے جو array کی ویلیو کے برابر * پرنٹ کرواتی ہے۔جب آپ اس پروگرام کو رن کریں گے تو اس کی آؤٹ پٹ یہ ہوگی۔

array index	value	Magic Histogram
0	15	* * * * * * * * * * * * * * *
1	8	* * * * * * * *
2	2	* *
3	7	* * * * * * *
4	11	* * * * * * * * * * *

5	5	* * * * *
6	3	* * *

یہ پروگرام اتنے * پرنٹ کرے گا جتنی اس ارے انڈیکس کی ویلیو ہوگی۔

Linear سرچ:

آپ نے کمپیوٹر کی تعریف میں یہ پڑھا ہوگا کہ کمپیوٹر معلومات سٹور کرنے اور دوبارہ حاصل کرنے کے لئے استعمال ہوتا ہے۔ خواہ ڈیٹا کسی بھی فارم میں ہو اور کسی بھی آرڈر میں محفوظ کیا گیا ہو مثلاً اریز وغیرہ۔ اس کے علاوہ آپ (پروگرامر) ایسی اریز استعمال کر رہے ہیں جس میں کافی ڈیٹا سٹور ہے اور ایسا اکثر ہوتا ہے اور ہر معلومات کی ایک پرائمری کی (Primary Key) ہوتی ہے جو ہر ریکارڈ کو ایک دوسرے سے مختلف کرتی ہے اور یہ کیز (Keys) ایک جیسی نہیں ہوسکتیں یعنی ان کی ویلیو منفرد ہوتی ہے۔ اس کے لئے پروگرام کو ہر ریکارڈ چیک کرنا ہوتا ہے کہ پروگرام میں کہیں کسی ریکارڈ کی تمام معلومات ایک جیسی تو نہیں ہیں کم از کم ایک فیلڈ تو منفرد ہونا چاہئے۔ ایک ارے کے مخصوص عنصر کے ڈھونڈنے کے عمل کو سرچنگ کہتے ہیں۔ سرچنگ کے کئی طریقے ہو سکتے ہیں۔ مگر آپ یہاں پر دو اہم طریقے پڑھیں گے جو یہ ہیں۔

- Linear سرچ
- Binary سرچ

سرچ کا ایک آسان طریقہ یہ ہے کہ کسی بھی اوبجیکٹ کو تلاش کرنے کے لئے ارے کے شروع سے ہر ایلیمنٹ کو یکے بعد دیگرے (ایک کے بعد دوسرا) چیک کیا جائے اور یہ عمل اس وقت تک جاری رہے جب تک مطلوبہ اوبجیکٹ مل نہیں جاتا۔ یعنی ارے کے ہر ایلیمنٹ کو سرچ کے ساتھ ملایا جاتا ہے۔ اس کو Linear سرچ کہتے ہیں۔ آیئے C++ میں اس کا طریقہ دیکھتے ہیں۔

مثال نمبر 3.4 Linear سرچ

```
#        //libraries
void search(int& found, int& index, int arr[], int a, int ans);
main( )
   {
   int count[]={12,81,2,99,5,16,18,72};
   int ans, found, location;
do  {
   cout <<"Enter number to find";
   cin >>ans;
   search(found,location,count,8,ans);         //function calling
   if(found)
    {
    location--;          //(boolean)found yes or no
    cout <<ans <<"is at index#" <<location <<endl;
    else
    cout <<ans <<"is not found!";
    }      while(ans !=0);
```

```
getch( );
}        //searching function
void search(int& found, int& index, int arr[], int a, int ans)
{
found=index=0;
while(!found && index<a)   {
found=(arr[index++]==ans);
}        //end of for loop
}        //end of search
```

اس پروگرام میں جب بھی اپنا مطلوبہ نمبر لکھیں گے تو اس کا ہر دفعہ arr[index] کے ساتھ موازنہ کیا جائے گا۔ اگر وہ نمبر مل جائے گا تو اس کا سب سکرپٹ نمبر ڈسپلے کر دیا جائے گا یعنی کہ اس انڈیکس کا نمبر یہ ہے ورنہ یہ ڈسپلے ہو جائے گا کہ نمبر نہیں ملا۔ جب آپ اس پروگرام کو ایگزیکیوٹ کریں گے تو یہ اس طرح آؤٹ پٹ ڈسپلے کرے گا۔

```
Enter number to find  99
99 is at index # 3
Enter number to find  8
8 is not found!
Enter number to find  0
0 is not found!
```

اور اس کے ساتھ ہی پروگرام ختم ہو جائے گا۔ ہم نے لوپ میں while(ans!=0) لکھا ہے یعنی اس وقت تک یہ لوپ ایگزیکیوٹ ہوتا رہے جب تک یوزر صفر (0) ان پٹ نہیں دیتا۔

Binary Search: بائنری سرچ:

یہ بہت تیز سرچنگ ٹیکنالوجی ہے۔ یہ ارے کو دو حصوں میں تقسیم کر دیتی ہے اور پھر ان دو حصوں میں سے مطلوبہ نمبر تلاش کرتی ہے۔ یہ ٹیکنالوجی کس طرح کام کرتی ہے آیئے اس کی ایک مثال دیکھتے ہیں۔

مثال نمبر 3.5 بائنری سرچ ٹیکنالوجی

```
#     //header files
void bsearch(int& found, int& index, int arr[], int a, int ans);
void main( )
{
clrscr( );
int count[]={12,10,81,9,29,71,20,0};
int temp, answer, location;
do  {
```

```
cout <<"Enter a number to find";
bsearch(answer, location, count, 8, temp);
if(answer)          //number found or not
cout <<temp <<"is found at index#" <<location <<endl;
else
cout <<temp <<"is not found";
while(temp !=0);
//Binary searching function
void bsearch(int& found, int& index, int arr[], int a, int ans)
{
int temp1=0, temp2=a-1;
found=0;
while(!found& temp1<=temp2)
{
index=(temp1+temp2)/2;     //Centeral point
found=(arr[index]==ans);
if(arr[index<ans)
temp=index+1;
else
temp2=index-1;
}
}
```

اریز کو ترتیب دینا:

C++ لینگوئج میں اسے Sorting کہتے ہیں۔ اس کا مطلب ہوتا ہے کہ اریز ڈیٹا کو ایک ترتیب سے ظاہر کرنا اور یہ بہت اہم کام ہے۔ مثلاً سکول میں تمام طالبعلموں کو ان کے رولنمبر کی ترتیب سے لکھا جاتا ہے، بنک اپنے تمام چیکس کو اکاؤنٹ نمبر کے مطابق ترتیب دیتا ہے غرض کہ تمام کمپنیاں اپنے ڈیٹا کو ایک خاص ترتیب سے لکھتی ہیں۔

C++ میں آپ ڈیٹا کو ترتیب کیسے دے سکتے ہیں؟ آیئے اس کے لئے ایک پروگرام لکھتے ہیں۔

مثال نمبر 3.6 Sorting Array

```
#header files
void main( )
{
clrscr( );
```

```
int count1, count2, temp;
const int size=8;
int arr[size]={1,9,3,7,11,5,0,13};
cout <<"Here is original array" <<endl;
for(count1=0; count1<size; count1++)
cout <<arr[count1];
for(count1=0; count1<size-1; count1++)
{
for(count2=0; count2<size-1; count2++)
{
if(arr[count2]>arr[count2+1])
{
temp=arr[count2];
arr[count2]= arr[count2+1];
arr[count2+1]= temp;
}       //end if
}       //end for
}       //end for
cout <<"Here is sorting Data" <<endl;
for(count1=0; count1<size; count1++)
count <<arr[count1];
getch( );
}
```

اس پروگرام میں سب سے پہلے a[0] کا a[1] سے موازنہ کیا جائے گا پھر a[1] کا a[2] سے اس کے بعد a[2] کا a[3] سے۔ اس طرح یہ a[8] تک جائے گا اور ہر دفعہ یہ نمبر کو ترتیب سے دے گا۔ مثلاً اگر a[1] نمبر a[2] سے بڑا ہوگا تو یہ اس کو a[0] پر سٹور کرے گا۔ اس کے لئے ہم نے for لوپ استعمال کی ہے اور اس میں ان نمبرز کو swap کروایا ہے۔

```
temp=arr[count2];
arr[count2]=arr[count2+1];
arr[count2+1]=temp;
```

یعنی arr[0] temp ویری ایبل میں سٹور ہو جائے گا اس کے بعد arr[0] میں 1 جمع کیا ہے۔ یوں arr[1] میں موجود ویلیو arr[0] میں آ جائے گی اور arr[0] والی ویلیو temp میں تھی وہ arr[1] میں آ جائے گی۔ یوں یہ پروگرام کام کرے گا جب آپ اوپر والی مثال 3.6 کو ایگزیکیوٹ کریں گے تو اس کا رزلٹ یہ ہوگا۔

```
Here is original array
```

```
1  9  3  7  11  5  0  13
Here is original array
0 , 1 , 3 , 5 , 7 , 9 , 11 , 13
```

یہ ٹیکنالوجی یا طریقہ ہم نے مثال نمبر 3.6 میں استعمال کیا ہے۔ یہ bubble sort یا sink sort کہلاتا ہے۔ اب ہم ایک ایسا پروگرام بنائیں گے جو ارے میں موجود نمبرز کو کاؤنٹ کریں گے۔ یعنی کونسا نمبر کتنی دفعہ ارے میں لکھا گیا ہے۔ یہ صرف آپ کی پریکٹس کے لئے ہے تاکہ آپ کو اریز پر زیادہ سے زیادہ کمانڈ حاصل ہو۔

مثال نمبر 3.7 اریز ویلیو کاؤنٹ کرنا

```
//#Header files
void main( )    {
    clrscr( );
    const int size=34, countsize=10;
    int array[size]=  {0,1,3,2,9,5,7,2,3,4,5,6,7,8,9,2,3,4,5,6,7,
                       8,9,10,8,3,2,1,4,5,3,2,10};
    int count[countsize]={0};
    for(int i=0; i<size; i++)
    ++count[array[i]];
    cout <<"Elements" <<setw(13) <<"Repitition" <<endl;
    for(int i=1; i<contsize; i++)
    cout <<i <<setw(13) <<count[i] <<endl;
    getch( );
    }
```

اس پروگرام کی آؤٹ پٹ یہ ہوگی۔

Elements	Repitition
0	2
1	3
2	5
3	5
4	3
5	4
6	2
7	3
8	3
9	3

10 1

ملٹی پل سب سکرپٹس ارے:

آپ نے میتھس میں ملٹی پل اریز کے بارے میں پڑھا ہوگا۔ یہ ڈیٹا کو ٹیبل میں قطاروں اور کالمز کی شکل میں ڈسپلے کروانے کے لئے استعمال ہوتی ہے۔ C++ میں بھی یہ اسی کام کے لئے استعمال کی جاتی ہیں۔ اس کے لئے ہم دو سکرپٹس واضح کرتے ہیں۔ پہلا قطار کو ظاہر کرنا ہے جبکہ دوسرا کالم کے لئے استعمال ہوتا ہے۔ C++ میں ملٹی پل سب سکرپٹس ارے کیسے بناتے ہیں مثلاً

```
int array[2][3]={{2,4,8},{1,3,5}};
```

اس میں دو قطاریں اور تین کالم ہیں۔ اب اس کو یوں ڈسپلے کروایا جائے گا۔

```
for(int i=0; i<2; i++)
{
for(int j=0; j<3; j++)
cout <<array[i][j];
}
```

اس میں پہلا لوپ قطاروں کو کنٹرول کرتا ہے جبکہ کالمز کے لئے دوسرا لوپ لکھا گیا ہے۔ اس کی آؤٹ پٹ یوں ہوگی۔

```
2   4   8
1   3   5
```

ابھی آپ نے دوسمتی ارے کے بارے میں پڑھا ہے لیکن اوپر مثال میں ہم نے صرف ارے اینی شلائیز کروائی ہے اور بعد میں ڈسپلے کروائی ہے۔ آپ اریز کے ساتھ عمل بھی کر سکتے ہیں۔ آیئے اس کے لئے اگلی مثال 3.7 دیکھتے ہیں۔ جس میں دو میٹرکس کو جمع کر کے تیسری میٹرکس میں محفوظ کروایا گیا ہے۔

مثال نمبر 3.8 تین سمتی ارے

```
//header files
void main(void)
   {
   clrscr( );
   int array[3][3],array2[3][3], array3 [3][3];
   int a, a1;
   cout <<"Enteries for 1st Matrix" <<endl;
   for(a=0; a<3; a++)
   for(a1=0; a1<3; a1++)
   {
   cout [<<" " <<a <<"]" <<"[" <<a1 <<"]=";
   cin >>array1[a][a1];
   }
```

```
cout <<"Second Matrix" <<endl;
for(a=0; a<3; a++)
for(a1=0; a1<3; a1++)
{
cout <<"[" <<a <<"]" <<"[" <<a1 <<"]=";
cin >>array2[a][a1];
}
cout <<"Resulting Matrix:" <<endl;
for(a=0; a<3; a++)
for(a1=0; a1<3; a1++)
{
array3[a][a1]=array1[a][a1]+array3[a][a1];
cout <<"[" <<a <<"]" <<"[" <<a1 <<"]=" <<array2[a][a1] <endl;
}
getch( );
}
```

اب جب آپ اس پروگرام کو کمپائل کرنے کے بعد ایگزیکیوٹ کریں گے تو پہلے یہ ان پٹ لے گا پھر دو اریز کو جمع کرے گا اور جواب array3[a][a1] میں سٹور کرے گا۔ اس پروگرام کی آؤٹ پٹ یہ ہوگی۔

```
Enteries for 1st Matrix:
[0] [0] = 1
[0] [1] = 2
[0] [2] = 3
[1] [0] = 1
[1] [1] = 2
[1] [2] = 3
[2] [0] = 4
[2] [1] = 5
[2] [2] = 6
Enteries for Second Matrix:
[0] [0] = 7
[0] [1] = 8
[0] [2] = 9
[1] [0] = 45
```

```
[1] [1] = 46
[1] [2] = 25
[2] [0] = 12
[2] [1] = 21
[2] [2] = 0
Resulting Matrix:
[0] [0] = 8
[0] [1] = 40
[0] [2] = 12
[1] [0] = 46
[1] [1] = 48
[1] [2] = 28
[2] [0] = 16
[2] [1] = 26
[2] [2] = 6
```

اس مثال میں ہم نے ہر کالم کی ان پٹ الگ سے لی ہے اور ساتھ ہی یہ بھی بتایا ہے کہ آپ کون سے نمبر کے لئے ان پٹ دے رہے ہیں تا کہ آپ کو پتہ چل سکے کہ سٹورایبل اریز میں کس طرح ان پٹ لی جاتی ہے اور کیسے اریز اسے سٹور کرتی ہے۔ آپ اس کے علاوہ نارمل ان پٹ بھی لے سکتے ہیں مگر وہ تھوڑا سا مشکل ہوتا ہے کہ ان پٹ کیسے دینی ہے۔ آیئے اب ایک اور مثال دیکھتے ہیں۔ جس میں آپ ارے فنکشن کو پاس کرنے کا طریقہ پڑھیں گے یعنی ارے کس طرح بطور فنکشن پیرامیٹر استعمال کی جا سکتی ہے۔

مثال نمبر 3.9 ارے کا استعمال

```
//header files
const int employee=3            //global variable
const int sale=4;               //global variable
int control=0;                  //funtrion to get average
float average(int temp[], int count)
    {
    int result=0;
    for(int i=0; i<count; i++)
    result+=temp[i];
    cout <<"\n Total of Row:" <<control<< "=" <<result <<endl;
    control+=1;           //increment 1 and again assign to control
    return (float) result/count;       //return average
    }              //end of function
```

```
void output(int sales[][sale], int dept, int test)
    {
    cout <<endl <<setw(22) <<"[0] [1] [2] [3]";
    for(int i=0; i<dept; i++)
    {
    cout<< "\n[" <<i <<"]" <<setw(7);
    for(int j=0; j<test; j++)
    cout <<setiosflags(ios::left) <<setw(6) <<sales[i][j];
    }          //end for loop
    }          //end function
void main(void)
    {
    clrscr( );
    int array[employee][sales];
    cout <<"Enter values as 4 number in one Row" <<endl;
    cout <<"And Then press enter" <<endl;
    for(int i=0; i<employee; i++)
    for(int j=0; j<sale; j++)
    cin >>array[i][j];                   //for input
    cout <<endl;
    output(array, employee, sale);       //function call
    for(int i=0; i<employee; i++)
    cout <<"\n Average of" <<i <<"=" <<setprecision(2)
                          <<average(array[i], sale);
    cout <<endl;
    getch( );
    }
```

جب آپ اس پروگرام کو ایگزیکیوٹ کریں گے تو یہ رزلٹ ڈسپلے ہوگا۔

```
Enter values as 4 number in one Row
And Then press enter
24  12  3   6
4   51  7   8
61  31  33  25
```

```
        [0]     [1]     [2]     [3]
    [0] 24      12      3       6
    [1] 4       51      7       8
    [2] 61      31      33      25
    Total of Row 0 = 45
    Average of 0 = 11.25
    Total of Row 1 = 70
    Average of 1 = 17.5
    Total of Row 2 = 150
    Average of 2 = 37.5
```

اس پروگرام میں فنکشن کو ارے پاس کی گئی ہے۔ فنکشن ( )output آپ کے تحریر کردہ ڈیٹا کی دوبارہ آؤٹ پٹ شو کرتا ہے کہ آپ نے کس سب سکرپٹ پر کونسی ویلیو دی ہے۔ اس کے علاوہ ایک اور فنکشن ( )average کا ہے جو اس ارے کی اوسط ریٹرن کرتا ہے۔ اور اس کے علاوہ یہ ہر قطار کا رزلٹ (جمع) بھی شو کرواتا ہے۔ اس فنکشن میں ایک لائن یہ ہے۔

```
control+=1;
```

یہ ہر لائن نمبر الگ سے پرنٹ کروانے کے لئے استعمال کیا گیا ہے۔ اس کے علاوہ ( )main فنکشن میں ہم نے ایک فنکشن setprecision(2) استعمال کیا ہے جو اوسط کی ریٹرن کردہ ویلیو میں اعشاریہ کے بعد والی ویلیو کو کنٹرول کرتا ہے اور صرف دو نمبرز 11.25 ریٹرن کرتا ہے۔

سٹرنگز: (Strings):

سٹرنگ میموری میں بامعانی کریکٹرز کی ترتیب کو کہتے ہیں جس کا اختتام Null('\o') کریکٹر سے ہوتا ہے یا سٹرنگ کریکٹر کی ایک ارے کو کہتے ہیں اور اس کے آخر پر ('\o') کریکٹر ہوتا ہے۔ اس کا مطلب یہ ہے کہ سٹرنگ کی لمبائی ہمیشہ اس میں موجود کریکٹرز سے ایک زیادہ ہوگی کیونکہ اس میں ایک Null کریکٹر لازمی آنا ہوتا ہے۔ آپ سٹرنگ یوں ڈیکلیر اور انیشلائیز کر سکتے ہیں۔

```
char str[]="Sikandar";
cout <<str;
```

آیئے اس کی ایک مثال لکھتے ہیں۔

مثال نمبر 3.10 سٹرنگ

```
void main(void)
    {
    char str[]="jutt";
    for(int i=0; i<5; i++)
    cout <<"str[" <<i <<"]" <<str[i] <<endl;
    getch( );
    }
```

اب جب یہ پروگرام ایگزیکیوٹ کیا جائے گا تو یہ رزلٹ ڈسپلے کرے گا۔

```
str [0] = j
str [1] = u
str [2] = t
str [3] = t
str [4] = ' '
```

سٹرنگ لائبریری:

اس کے علاوہ ++C سٹرنگ کے کچھ پہلے سے بنے ہوئے فنکشنز کی سہولت بھی دیتی ہے۔ جس میں سٹرنگ ڈیٹا کاپی کرنا، سٹرنگ کی لمبائی معلوم کرنا وغیرہ شامل ہیں۔ اگر آپ سٹرنگ کا کوئی بھی فنکشن استعمال کرنا چاہتے ہیں تو اس کے لئے آپ کو ++C کی سٹینڈرڈ لائبریری <string.h> استعمال کرنا ہوگی۔ نیچے ٹیبل میں چند اہم اور مفید سٹرنگ فنکشنز کی لسٹ ترتیب دی گئی اور ان کا استعمال آپ بعد میں دیکھیں گے۔

فنکشن	وضاحت
strcat( )	strcat(temp1,blank) یہ temp1 میں blank کی ویلیو سٹور کروائے گا۔
strchr( )	strchr(string,c) یہ اس بات کی نشاندہی کرے گا کہ ویری ایبل c کی ویلیو سٹرنگ میں کس پوزیشن پر ہے اور اگر c کی ویلیو سٹرنگ میں شامل نہیں ہوگی تو یہ Null ریٹرن کرے گا۔
strcmp( )	strcmp(s1,s2) یہ s1 کا s2 سے موازنہ کرتا ہے۔
strcpy( )	strcpy(s1,s2) یہ s2 کو s1 میں کاپی کرنے کے لئے استعمال ہوتا ہے۔
strlen( )	strlen(string) یہ سٹرنگ کی لمبائی معلوم کرنے کے لئے استعمال کیا جاتا ہے۔
strncpy( )	strncpy(s1,s2,3) اس میں یہ بات واضح کر سکتے ہیں کہ s2 سٹرنگ کا کتنا حصہ s1 میں کاپی ہو۔
strncat( )	strncat(s1,s2,7) اس میں یہ بات واضح کرنی ہے کہ s2 کا کتنا حصہ s1 میں آئے۔
strncmp( )	strncmp(s1,s2,3) یہ s1 کے s2 سے بتائے گئے کریکٹرز کا موازنہ کرتا ہے۔
strpbrk( )	strpbrk(s1,s2) یہ s2 سٹرنگ سے پہلے s1 سٹرنگ میں پہلی موجودگی کو ظاہر کرتا ہے۔
strspn( )	strspn(str1,str2) یہ سٹرنگ 1 اور سٹرنگ 2 کی ویلیو چیک کرتا ہے اور سب سے پہلے جس انڈیکس نمبر پر ان کی ویلیو مختلف ہو وہ انڈیکس نمبر ریٹرن کرتا ہے۔
strstr( )	strstr(s1,s2) اس سے آپ یہ واضح کرتے ہیں کہ s2 کا کون سا کریکٹر یا ورڈ s1 کا حصہ ہے۔
strcmpi( )	strcmpi(s1,s2) یہ دو سٹرنگز کا موازنہ کرتا ہے لیکن یہ اس بات کا لحاظ نہیں رکھتا کہ دونوں سٹرنگز کا کیس کیا ہے یعنی یہ case sensitive نہیں ہے۔

آپ نے اوپر ٹیبل میں ++C کی <string.h> ہیڈر فائل کے چند اہم فنکشنز کے بارے میں مختصراً پڑھا۔ اب آگے آپ ان کا استعمال پڑھیں گے کہ یہ کس طرح کام کرتے ہیں۔

آیئے اب ایک پروگرام لکھتے ہیں جو یوزر سے دو ان پٹس لے گا اور ان کا آپس میں موازنہ کرے گا کہ کون سا سٹرنگ بڑا ہے یا چھوٹا ہے۔

مثال نمبر 3.11 کمپیرنگ ٹو سٹرنگز

```
//get( ) header file
```

```
#include<iostream.h>
#include<stdio.h>
#include<string.h>
#include<conio.h>
void main(void)
    {
    clrscr( );
    char a[10], a1[15];
    int x;
    cout <<"Enter First string:";
    gets(a);                //for string input with space
    cout <<"Enter Second string:";
    gets(a1);
    x=strcmpi(a,a1);        //compare two strings
                            //without case sensitive
    if(x==0)
    cout <<a <<": is equal to:" <<a1;
    else
    if(x>0)
    cout <<a <<":is greater than:" <<a1;
    else
    if(x<0)
    cout <<"Something is wrong Try again:";
    getch( );
    }
```

جب آپ اس پروگرام کو کمپائل کرنے کے بعد ایگزیکیوٹ کریں گے تو یہ آؤٹ پٹ حاصل ہوگی۔ یہ ان پٹ ہم نے فرضی دی ہے آپ کچھ بھی ان پٹ دے سکتے ہیں۔

```
Enter First string:    Welcome to Gojra
Enter Second string:   I Live in Gojra
Welcome to Gojra: is greater than: I live in Gojra
```

ہم نے اس پروگرام میں دو کریکٹر اریز [10]a اور [15]a لی ہیں اور یوزر سے ان پٹ ( )gets فنکشن سے لی ہے اس لئے کہ cin یوزر سے سپیس نہیں لیتا۔ پھر ہم نے ان دونوں سٹرنگز کا (strcmpi(a,a1 کی مدد سے موازنہ کروایا ہے اور اس کی ویلیو x ویری ایبل میں سٹور کی ہے کہ یہ دونوں برابر ہیں یا نہیں۔ ہم نے ( )strcmpi لکھا ہے یہ اس بات کا لحاظ نہیں کرتا کہ دونوں سٹرنگز کا حروفِ تہجی لکھنے کا سٹائل بھی ایک جیسا ہو۔ یعنی یہ دو سٹرنگز کا

موازنہ Case Sensitivity کے بغیر کرتا ہے۔ یعنی اگر آپ یہ ان پٹ دیتے ہیں۔

```
Enter First string:    HELLO
Enter Second string:   Hello
```

تو یہ دونوں کو equal شو کرے گا اس لئے کہ یہ case sensitive نہیں ہے لیکن اگر آپ چاہتے ہیں کہ ہمارا فنکشن case sensitive ہو تو پھر strcmp( ) فنکشن استعمال کریں۔ باقی سارا عمل وہی ہے صرف یہ لائن تبدیل کریں۔

```
strmp(a,a1)
```

سٹرنگ جمع کرنا:

آیئے اب سٹرنگ کے لئے ایسا پروگرام لکھتے ہیں جو دو سٹرنگز کو (جمع) کرے گا۔ ایک سٹرنگ کو دوسرے کے آخر پر لگائے (چسپاں کرے) گا اور ساتھ میں ان کی لمبائی بھی شو کروائے گا۔

مثال نمبر 3.12 Strcat اینڈ strlen فنکشن کا استعمال

```
//header files
void main(void)
{
 clrscr( );
 char temp1[]="Shoaib and";
 char temp2[]="Awais";
 cout<<"Before strcat function";
 cout <<"\n First string=" <<temp1 <<",[length]=" <<strlen(temp1);
 cout <<"\n Second string=" <<temp2 <<",[length]=" <<strlen(temp2);
 strcat(temp1, temp2);
 cout <<"\n after strcat function";
 cout <<"\n 1st string=" <<temp1 <<",[length]=" <<strlen(temp1);
 cout <<"\n 2nd string=" <<temp2;
 cout <<"\n Enter a string:";
 gets(temp1);
 cout <<"temp1 <<": is length=" <<strlen(temp1);
 getch( );
 }
```

اس پروگرام میں ہم نے دو کریکٹر اریز [ ]temp1 اور [ ]temp2 بنائی ہیں اور ان کی لمبائی واضح نہیں کی یعنی یہ ہم رن ٹائم پر جتنی مرضی لمبی کر لیں۔ پھر strcat( ) کی مدد سے ان کو اکٹھا کیا ہے اور ان کی لمبائی معلوم کی ہے اور بعد میں [ ]temp1 میں یوزر سے ان پٹ لی اور اس سٹرنگ کی لمبائی معلوم کی ہے۔ اس پروگرام کی آؤٹ پٹ یہ ہوگی۔

```
Before strcat function
```

```
First string = Shoaib and          ,[length]=10
second string = Awais              ,[length]=6
After strcat function
1st string = Shoaib and Awais ,[length]=16
2nd string = Awais
Enter a string Yasir and Sikandar
Sikandar and Yasir                 ,[length]=18
```

سٹرنگ کاپی کرنا:

آپ ایک سٹرنگ کو دوسرے سٹرنگ پر کاپی بھی کر سکتے ہیں۔اس کے لئے ++C کی سٹینڈرڈ لائبریری استعمال کی جاتی ہے۔آیئے اس کی ایک مثال دیکھتے ہیں۔اس میں ہم ++C کا سٹینڈرڈ فنکشن ( )strcpy استعمال کریں گے یہ پہلے سٹرنگ پر دوسرے سٹرنگ کو کاپی کرنے کے لئے استعمال کیا جاتا ہے۔

مثال نمبر 3.13 سٹرنگ کاپی کرنا

```
//header files
void main( )
  {
  clrscr( );
  char string1[30], string2[25];
  char choice;
  do
  {
  cout <<"Enter First String:";
  gets(string1);
  cout <<"\n Enter Second String:";
  gets(string2);
  cout <<"\n Before strcpy function:";
  cout <<"\n 1st string is:" <<string1;
  cout <<"\n 2nd string is:" <<string2;
  strcpy(string1, string2);
  cout <<"\n After strcpy function:";
  cout <<"\n 1st string:" <<string1;
  cout <<"\n 2nd string:" <<string2;
  cout <<"\n Do You want to continue?";
```

```
cin >>choice;
}
while(choice=='y'¦choice=='y');
}                              //end main
```

اس پروگرام کی آؤٹ پٹ کچھ یوں ہوگی۔

```
Enter 1st string:  Welcome to My book
Enter 2nd string:  Ecommerce
Before strcpy function
1st string: Ecommerce
2nd string: Ecommerce
Do You want to countinue? n
```

آپ نے دیکھا کہ اس میں دوسرا اسٹرنگ پہلے سٹرنگ پر کاپی کر دیا جاتا ہے اور پہلا سٹرنگ ختم ہو جاتا ہے۔ اس کے علاوہ ایک اور فنکشن strncpy() ہے یہ سٹرنگ 2 کا کچھ مخصوص ڈیٹا سٹرنگ کے مخصوص حصے پر کاپی کر دے گا اس کے لئے باقی پروگرام وہی رہے گا صرف یہ لائن تبدیل کریں۔

```
strncpy(string1, string2, 6);
```

اب فرض کریں آپ اسے ایگزیکیوٹ کرتے ہیں تو یہ ان پٹ مانگے گا آپ اسی یہ ان پٹ دیتے ہیں۔

```
Enter 1st string:  Hello How Are You
Enter 2nd string:  Thanks Fine
Before strncpy function
1st string: Thanks How Are you
2nd string: Thanks Fine
Do You want to countinue? n
```

آؤٹ پٹ پر غور کریں کہ اس نے سٹرنگ 1 کے پہلے 6 کریکٹرز پر سٹرنگ 2 کے چھ کریکٹرز چسپاں کر دیئے ہیں لیکن سٹرنگ میں کوئی تبدیلی نہیں آئی وہ بالکل ویسا ہی ہے۔

سٹرنگز کو اکٹھا کرنا:

آپ نے ایک سٹرنگ کو دوسرے سٹرنگ کے ساتھ ملانے کا طریقہ پہلے بھی دیکھا ہے لیکن اب ہم آپ کو اس سے تھوڑا سا مختلف طریقہ بتائیں گے۔ اس کے علاوہ آپ اس پروگرام میں strchr( ) فنکشن کا استعمال بھی دیکھیں گے۔ تو اس کے لئے ایک پروگرام لکھتے ہیں۔

مثال نمبر 3.14 strncat اور strchr فنکشن

```
//header files
void main( )
 {
 clrscr( );
 char string1[]="Yasir and";
```

```
char string2[]="Sikandar are best Friends";
int answer;
cout <<"Before strncat Function.";
cout <<"\n string1=" <<string1;
cout <<"\n string2=" <<string2;
strncat(string1, string2, 8);
cout <<"After strncat Function.";
cout <<"\n string1=" <<string1 <<",[length]=" <<strlen(string1);
cout <<"\n string2=" <<string2 <<",[length]=" <<strlen(string2);
cout <<endl;
cout <<"\n Use of strchr end strstr";
answer=(int) strchr(string1, 'd');
cout <<\n strchr(string1,'d') is at location=" <<ans-(int)string1;
cout <<"\n Enter string to find from string2:";
gets(string1);
answer=(int) strchr(string2, string1);
cout <<\n string1 <<"is at location=" <<ans-(int)string2;
getch( );
}
```

آیئے اب اس پروگرام کی آؤٹ پٹ دیکھتے ہیں۔

```
Before strncat Function
string1 = Yasir and
string2 = Sikandar are best Friends
After strncat Function
string1 = Yasir and Sikandar           ,[length] = 18
string2 = Sikandar are best Friends   ,[length] = 26
strchr(string1, 'd') is at location = 8
Enter string to find from string2: are
are is at location = 9
```

یہ اس پروگرام کی آؤٹ پٹ ہے اس میں ہم نے دو اریز لی ہیں اور پہلی سٹرنگ ارے کے آخر پر دوسری سٹرنگ ارے کا کچھ حصہ تحریر کروایا ہے۔ اس کے لئے ہم نے strncat(string1, string2, 8) فنکشن استعمال کیا ہے۔ یہ فنکشن سٹرنگ 2 کے کریکٹرز سٹرنگ 1 کے آخر پر لگے گا۔ یعنی آپ جتنی ویلیو درج کریں گے یہ اتنے کریکٹرز تحریر کرے گا۔ اس کے بعد ہم نے ایک فنکشن strchr( ) استعمال کیا ہے۔

```
answer = (int)strchr(string1, 'd');
```

یہ فنکشن string1 ارے میں سے d کریکٹر تلاش کرے گا۔ یہاں پر آپ دیکھیں گے کہ ہم نے ٹائپ کاسٹنگ کی ہے وہ اس لئے کہ یہ جو ویلیو ریٹرن کرے گا وہ نمبرز میں ہوگی۔ ہم نے اس کا رزلٹ بھی int ٹائپ کے ویری ایبل میں سٹور کروایا ہے۔ اگر آپ ایسا نہیں کریں گے تو یہ ایرر دے گا۔

```
Cannot convert char* to int
```

اس کے بعد ہم نے انڈیکس نمبر کو پرنٹ کروایا ہے۔

```
cout <<"strchr(string1, 'd') is at location=" <<ans-(int) string1;
```

یہاں پر ہم نے answer میں سے سٹرنگ 1 کو تفریق کیا ہے۔ اب سٹرنگ char ٹائپ کا ہے اس لئے اس کی ٹائپ کاسٹنگ کی ہے اور ایک اہم بات ( )strchr فنکشن صرف کریکٹر کو تلاش کرنا ہے۔ اگر آپ سٹرنگ میں سے سٹرنگ کو تلاش کرنا چاہتے ہیں تو اس کے لئے ہم نے یہ لائن لکھی ہے۔

```
answer=(int)strstr(string2, string1);
```

اس کے بعد ہم نے ٹائپ کاسٹنگ کی ہے اور اس میں ہم نے string2 میں سے string1 کو تلاش کرنا ہے اور سٹرنگ 1 آپ یوزر سے پوچھ سکتے ہیں۔ آپ یہ خود بھی لکھ سکتے تھے مگر ہم نے یہ سہولت یوزر کو دی ہے۔ اگر آپ خود لکھنا چاہتے ہیں تو یوں لکھئے۔

```
answer=(int)strstr(string2, "are");
```

اب یہ are کو string2 میں سے تلاش کر کے اس کا انڈیکس نمبر ریٹرن کرے گا۔

ایک بات ہم نے یہاں پر ٹائپ کاسٹنگ کی ہے۔ آپ اس کے بغیر بھی یہ پروگرام بنا سکتے ہیں لیکن اس کے لئے آپ کو پوائنٹر استعمال کرنا ہوگا۔ پوائنٹر کیا ہے؟ یہ آپ آگے پڑھیں گے اور اس کی مدد سے آپ ایک پروگرام بھی بنائیں گے۔

مشق

سوال نمبر 1: اریز کیا ہیں اور یہ کس لئے استعمال ہوتی ہیں؟

سوال نمبر 2: ایک ارے بنائیں جس کا سائز 50 ہو اور آپ اس میں صرف 8 ویلیوز (Elements) تحریر کریں۔ اس کے بعد یوزر سے ایک ان پٹ لیں اور وہ اس ارے میں اس جگہ تحریر کریں جہاں اس کا درست آرڈر ہو۔ آپ کی ارے کے elements یہ ہونے چاہئیں۔ 500 , 450 , 391 , 312 , 298 , 272 , 261 , 112

اب فرض کریں کہ یوزر 150 نمبر لکھتا ہے تو یہ نمبر 112 اور 261 کے درمیان لکھا جانا چاہئے۔

سوال نمبر 3: ایک ایسا پروگرام تحریر کریں جو یوزر سے سٹرنگ ان پٹ لے اور پھر یہ اس کا الٹ آرڈر شو کرے یعنی اگر یوزر awais لکھتا ہے تو یہ siawa شو کرے اور ایسا ان پٹ میتھڈ استعمال کرنا ہے جو سٹرنگز میں سپیس کی سہولت فراہم کرتا ہو۔

سوال نمبر 4: ایک ایسا پروگرام تحریر کریں جس میں یوزر 3×3 اریز کے لئے ویلیو درج کرے اور پھر بعد میں ان کا مجموعہ اور اوسط ڈسپلے کروائیں۔ یہ کام آپ فنکشن میں کریں گے اور بطور ان پٹ 3×3 کی دو اریز یوزر سے لیں گے۔

سوال نمبر 5: ایک ایسا پروگرام بنائیں جو یوزر سے دو سٹرنگ لے اور ان کا موازنہ کرے کہ کیا دونوں سٹرنگز برابر ہیں یا نہیں اور ان کی لمبائی کیا ہے اور اس کے علاوہ ایک سٹرنگ کو دوسرے پر کاپی بھی کروائیں۔

سوال نمبر 6: اس پروگرام کی آؤٹ پٹ کیا ہوگی؟

```
int temp[40], count=0, option=0, total=0;
outer:
cout <<"Enter Number between[2-40]";
cin >>option;
if(option)=2&&option<=40)
    {
    for(cout=0; count<option; count++)
    cin >>temp[count];
    cout <<"\n";
    }                    //end for loop
    count=0;
    for(      ; count<option; count++)
```

```
{
total=total+temp[count];
cout <<"Total is=" <<total;
}
}                    //end if
else
{
cout <<"\n Wrong Entery:";
goto outer;
}
getch( );
}                    //end of main;
```

جوابات

❶: جواب

ارے ایسی ڈیٹا ٹائپ ہے جس میں ایک ہی نام کے ویری ایبلز سٹور کئے جاتے ہیں اور ارے میں میموری ایک خاص ترتیب سے اکٹھی جگہ گھیرتی ہے یا ارے ایسے اوبجیکٹس کی ترتیب کا نام ہے جن کی ڈیٹا ٹائپ ایک ہونی چاہئے یہ اوبجیکٹس ارے کے اجزاء کہلاتے ہیں۔ آپ اریز اس وقت استعمال کرتے ہیں جب آپ ایک قسم کا بہت زیادہ ڈیٹا میموری میں سٹور کروانا چاہتے ہیں۔

❷: جواب

```
#include<conio.h>
#include<iostream.h>
#include<iomanip.h>
void output(int[], int);
void insert(int[], int&, int);
main(void)
    {
    clrscr( );
    int end=8, temp;
    int array[50]={112,261,272,296,312,391,450,500};
    cout <<"Your Origional Array=" <<endl;
    output(array, end);
    cout <<"\n Enter Number:";
    cin >>temp;
    insert(array, end, temp);
    cout <<"\n Now number has been inserted:" <<endl;
    output(array, end);
    getch( );
    }
void insert(int x[], int& temp, int z)
    {
    int i=temp;
    for( ;i>0&& x[i-1]>z; i--)
    x[i]=x[i-1];
```

```
    x[i]=z;
    ++temp
    }
void output(int temp[], int x)
    {
    for (int i=0; i<x-1; i++)
    cout <<temp[i] <<",";
    if(i+1)%12==0)
    cout <<endl;
    }
    cout <<temp[x-1] <<endl;
    }
```

جواب: 3

```
#include<conio.h>
#include<iostream.h>
#include<stdio.h>
#include<string.h>
void reverse(char[]);
const int size=75;
char temp;
main(void)
    {
    clrscr( );
    char string[size];
    cout <<"Enter a String:";
    gets(string);                  //gets string form user
    reverse(string);
    cout <<"New modified string is:" <<string <<endl;
    cout <<"Enter String:";
    cin.get(string, size);         //cin.get(string,size);is also
                                     used to get string
    reverse(string);
```

```
cout <<"New modified string is:" <<string <<endl;
getch( );
}
void reverse(char a[])
{
int length=strlen(a);
for(int i=0; i<length/2; i++)
{
temp=a[i];
a[i]=a[length-i-1];
a[length-i-1]=temp;
}
}
```

: جواب

```
#include<conio.h>
#include<iostream.h>
#include<iomanip.h>
void average(void);                    //function declaration
int array1[3][3], array2[3][3];        //global array and variables
double sum[3][3];
in i,j;
void main(void)
{
clrscr( );
cout<< "\n First Matrix: \n";
cout <<"Enter 3 values in one row then press enter:" <<endl;
for (i=0; i<3; i++)
for (j=0; j<3; j++)
{
cin >>array1[i][j];                    //1st matrix input
cout <<"\n Second Matrix: \n;
cout <<"Enter 3 values in one row then press enter:" <<endl;
for(i=0; i<3; i++)
```

```
    for(j=0; j<3; j++)
    {
    cin >>array2[i][j];
    }                                       //2nd matrix input
    result( );                              //function calling
    average( );
    getch( );
    }
void result(void)                           //function defination
    {
    cout <<"\n Sum of Two Matrixes:" <<endl;
    for(i=0; i<3; i++)
    {
    cout <<endl;
    for(j=0; j<3; j++)
    {
    sum[i][j]=array1[i][j]+array2[i][j];
    cout <<setw(8) <<sum[i][j];
    }
    }                                       //end of for loop
    }                                       //end of function
void average(void)                          //function definition
    {
    cout <<"\n\n Average Matrix:" <<endl;
    for(i=0; i<3; i++)
    {
    cout <<endl;
    for(j=0; j<3; j++)
    cout <<setw(8) <<(sum[i][j]/=2);
    }
    }
```

● : جواب

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
#include<stdio.h>
void main(void)
   {
   clrscr( );
   char a[10], a1[15];
   cout <<"\n enter first string:";
   gets(a);
   cout <<"\n length of" <<a <<":" <<strlen(a);
   cout <<"\n length of" <<a1 <<":" <<strlen(a1);
   cout <<endl;
   if(strcmp(a,a1)==0)
   cout <<a <<"==" <<a1;
   else
   cout <<a <<"!=" <<a1;
   cout <<"\n copying Strings" <<endl;
   cout <<"\n New First String:" <<strcpy(a,a1);
   getch( );
   }
```

● : جواب

```
Enter Number between(2-40) 3
Enter 3 values : 6
8
5
Total is : 19
```

باب نمبر 4

پوائنٹرز

اس باب میں آپ پوائنٹر کے بارے میں پڑھیں گے۔ یہ C اور ++C کا ایک اضافی فیچر ہے کیونکہ یہ زیادہ تر کمپیوٹر لینگویجز میں شامل نہیں ہے مثلاً جاوا، پاسکل، بیسک وغیرہ۔ آپ سوچ رہے ہوں گے کہ یہ پوائنٹرز کیا ہیں اور یہ کیوں استعمال ہوتے ہیں۔ تو پوائنٹر مندرجہ ذیل کام سرانجام دینے کے لئے استعمال ہوتا ہے۔

- ارے عناصر کو ایکسیس کرنا
- فنکشن کو آرگومنٹ پاس کرنا تاکہ آپ انہیں رن ٹائم پر تبدیل کرسکیں
- فنکشنز کو اریز اور سٹرنگز پاس کرنا
- سسٹم کی میموری کا ایڈریس معلوم کرنا

ان تمام باتوں کے پیش نظر ہم نے اس باب میں پوائنٹرز کا تفصیلی ذکر کیا ہے۔ آپ پوائنٹر سے متعلقہ یہ عنوانات پڑھیں گے۔

- تعارف
- ریفرنسز
- پوائنٹر
- پوائنٹر اینڈ اریز
- ریفرنس ریٹرن کرنا
- فنکشن پوائنٹر
- کریکٹر اور سٹرنگ فنکشنز
- نیو آپریٹر
- ڈیلیٹ آپریٹر
- پوائنٹرز ٹو پوائنٹر
- اریز
- مشق

تعارف:

جب ایک ویری ایبل ڈیکلیر کیا جاتا ہے تو اس کے ساتھ تین اہم کلاز ز منسلک ہوتے ہیں مثلاً اس کا نام یعنی ویری ایبل کا نام، اس کی ٹائپ کیا ہے؟ اور اس کا ایڈریس کیا ہے؟ ایڈریس سے مراد یہ ہے کہ یہ میموری میں کس جگہ سٹور ہے۔ مثلاً

```
char temp;
```

اس ویری ایبل کا نام temp ہے۔ اس کی ڈیٹا ٹائپ کریکٹر ہے اور اس کے علاوہ میموری میں اس ویری ایبل کی ویلیوکس لوکیشن پر سٹور ہوگی یعنی اس کا میموری ایڈریس۔ فرض کریں کہ اس کا ایڈریس 0×2|AB00|1 ہے۔ یہ ویلیو Hexadecimal میں ہے اور آپ کی میموری میں یہ نمبر دیئے گئے ہیں۔ کوئی بھی ویری ایبل میموری میں یوں سٹور ہوتا ہے۔

```
        int temp;
        0X1FFOE15
int   [          ]
        temp
```

یہ باکس ویری ایبل کو میموری میں سٹور کرنے کا طریقہ بتا رہا ہے اس میں باکس کے بائیں طرف ڈیٹا ٹائپ ہے اور باکس کے نیچے ویری ایبل کا نام ہے اور اوپر کی طرف ویری ایبل کا میموری میں ایڈریس ہے اب اگر آپ اس کو کوئی ویلیو دیتے ہیں تو وہ اس باکس میں محفوظ ہوگی۔ مثلاً فرض کریں ہم نے اس میں 12 سٹور کروایا ہے تو وہ اس کے اندر لکھا ہوگا۔ کسی بھی ویری ایبل کی ویلیوکس طرح پرنٹ کرواتے ہیں اس سے آپ بخوبی واقف ہیں۔ یعنی cout<<temp; لکھا جائے گا۔ اسی طرح آپ کسی بھی ویری ایبل کا میموری میں موجود ایڈریس بھی معلوم کر سکتے ہیں۔ اس کے لئے '&' علامت استعمال کی جاتی ہے اس کو ایڈریس آپریٹر بھی کہتے ہیں اور کسی بھی ویری ایبل کا میموری ایڈریس معلوم کرنے کے لئے آپ اس کو یوں لکھتے ہیں۔

```
cout <<& temp;
```

آیئے اس کے لئے ایک چھوٹا سا پروگرام لکھتے ہیں۔

مثال نمبر 4.1 ویری ایبل کا ایڈریس معلوم کرنا

```
//header files
void main(void)
  {
  int temp=12;
  cout <<"Value of temp" <<temp;
  cout <<"\n Address of temp=" <<&temp;
  getch( );
  }
```

اب جب آپ اس پروگرام کو ایگزیکیوٹ کریں گے تو اس کی آؤٹ پٹ یہ ہوگی۔

```
Value of temp = 12
Address of temp = oxIffacfId
```

ریفرنسز: (References):

ایک ریفرنس اصل میں کسی دوسرے ویری ایبل کا ایک مترادف ہوتا ہے اس کو پروگرامنگ کی زبان میں کسی دوسری ویری ایبل کا Alias کہتے ہیں اور آپ کسی بھی ویری ایبل کا Alias اس کو ڈیکلیر کرتے وقت بناتے ہیں۔ اس کے لئے اس کے ساتھ ایڈریس آپریٹر (&) استعمال کیا جاتا ہے۔ یہ کس طرح کام کرتا ہے آیئے اس کو ایک مثال کی مدد سے سمجھتے ہیں۔

مثال نمبر 4.2 ریفرنس ویری ایبل ڈیکلیر کرنا

```
void main(void)
  {
  int temp=21;
  int& ref=temp;                    //ref is refrence for temp
  cout <<"temp=" <<temp <<",ref=" <<ref <<endl;
  temp+=4;
  cout <<"temp=" <<temp <<",ref=" <<ref;
  ++ref;
  cout <<"\n temp=" <<temp <<",ref=" <<ref;
  getch( );
  }
```

اس پروگرام کی آؤٹ پٹ پر غور کریں کہ اس میں کس بات کی نشاندہی کی جارہی ہے۔

```
temp = 21 , ref = 21
temp = 25 , ref = 25
temp = 26 , ref = 26
```

اوپر پروگرام میں ہم نے دو ایڈنٹیفائر مختلف ناموں کے لئے ہیں لیکن اصل میں یہ دونوں ایک ہی ویری ایبل کے الگ الگ ایڈنٹیفائر ہیں اور آپ نے آؤٹ پٹ میں دیکھا کہ ہم ایک ایڈنٹیفائر میں تبدیلی کرتے ہیں اور دوسرا ایڈنٹیفائر خود بخود تبدیل ہو جاتا ہے اور اسے دوسرے کی ویلیو بائی ڈیفالٹ آسائن کردی جاتی ہے۔

آپ حیران ہوں گے کہ آخر یہ کس طرح ممکن ہے؟ تو آیئے اس مثال کی مدد سے اس کی لاجک سمجھنے کی کوشش کرتے ہیں۔

مثال نمبر 4.3 ریفرنس ویری ایبلز کا ایڈریس معلوم کرنا

```
void main(void)
  {
  clrscr( );
  int temp=21;
  int& ref=temp;
  cout <<"temp=" <<temp <<"ref=" <<ref;
  cout <<"\n Memory Addresses";
```

```
cout <<"\n & temp=" <<& temp <<",& ref" <<& ref;
getch( );
}
```

اب اس کو اگز یکیوٹ کریں۔ ہمارے پاس اس کی آؤٹ پٹ کچھ یوں آتی ہے۔

```
temp = 21 , ref = 21
Memory Addresses
& temp = 0x|FFE04D , ref 0x|FFE04D
```

آپ نے دیکھا کہ دونوں ایڈنٹیفائرز کا میموری ایڈریس بھی ایک ہی ہے۔یعنی یہ میموری میں یوں سٹور ہیں۔

```
      int temp;
      0X11FFE04D
int  [   21   ]
      int
```

یعنی میموری میں ویلیو 21 صرف ایک دفعہ سٹور کی گئی ہے اور temp اور ref کا میموری میں ایک ہی ایڈریس ہے۔ کانسٹنٹ ایڈنٹیفائر کی طرح ریفرنس ایڈنٹیفائر بھی جب ڈیکلیر کیا جاتا ہے تو اسے اسی وقت اپنی شیلائیز کرنا ضروری ہوتا ہے۔

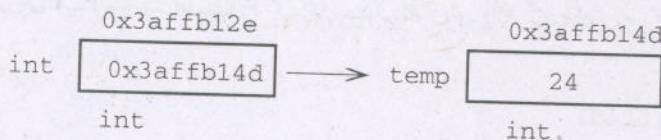
پوائنٹر :

ایک ویری ایبل اپنی ویلیو کو ڈائریکٹ ریفرنس کرتا ہے جبکہ ایک پوائنٹر اپنی ویلیو کو ان ڈائریکٹ ریفرنس کرتا ہے یعنی ویری ایبل میں اس کی ویلیو سٹور ہوتی ہے جبکہ پوائنٹر میں ایک ویری ایبل کا میموری ایڈریس ہوتا ہے اور یہ ایڈریس پھر متعلقہ ویلیو کی نشاندہی کرتا ہے دوسرے ویری ایبلز کی طرح پوائنٹرز بھی استعمال کرنے سے پہلے ڈیکلیر کرنا ضروری ہوتے ہیں۔

```
int * temp ; float * PI;
```

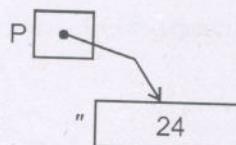
اس کو آپ یوں پڑھیں گے کہ یہ ایک فلوٹ ویری ایبل کے لئے پوائنٹر ہے یا ایک نمریک ویلیو کے لئے پوائنٹر ہے۔ پوائنٹر کس طرح کام کرتا ہے آیئے اس کی ایک مثال دیکھتے ہیں۔

مثال نمبر 4.4 پوائنٹر کا استعمال

```
//header files
void main( )
{
int temp=24;
int*ptr=& temp;          //ptr holds the address of temp
cout <<"temp=" <<temp <<",& temp=" <<& temp <<",prt="
                                          <<ptr <<endl;
cout <<"& prt=" <<& prt;
getch( );
}
```

اس پروگرام میں ایک ویری ایبل temp لیا ہے جس کی ویلیو 24 ہے اور ایک انٹ ٹائپ کا پوائنٹر ٹو ptr لیا ہے۔ اور اس میں ہم نے & temp سٹور کیا ہے اور بعد میں کچھ پرنٹ کروایا ہے جو اس پروگرام کی آؤٹ پٹ ہوگی۔

```
temp = 24 , & temp=0x3ffb14d , p=0x3affb14d
& ptr=0x3affb12e
```

آپ نے دیکھا کہ & temp اور p کی ویلیو 0x3ffb14d (ایک جیسی) ہے یعنی کہ temp ویری ایبل کا ایڈریس ptr میں سٹور ہے۔ یہ میموری میں کہیں بھی یوں محفوظ ہوگا۔

```
          0x3affb12e                          0x3affb14d
int   | 0x3affb14d |  ------>  temp  |     24     |
          int                                 int
```

ویری ایبل ptr کو یہاں ہم پوائنٹر کہیں گے کیونکہ اس کی ویلیو میموری میں موجود کسی اور ویلیو کو پوائنٹ کرتی ہے۔ اس کو ہم int پوائنٹر کہیں گے کیونکہ یہ جس ویلیو کو پوائنٹ کر رہی ہے وہ int ہے اور ptr پوائنٹر کی ویلیو ایک ایڈریس ہے۔ یہ ایڈریس ہر کمپیوٹر پر مختلف ہوگا یعنی ہمارے پاس جو ایڈریس ہے ہوسکتا ہے آپ کے پاس ایسا ایڈریس نہ آئے۔ اب ہم ایک پروگرام لکھتے ہیں جس میں ہم پوائنٹر کی ویلیو ڈسپلے کروائیں گے۔

مثال نمبر 4.5 پوائنٹر ویلیو ڈسپلے کرنا

```
void main(void)
  {
  int temp=24;
  int* ptr=& temp;
  cout <<"*ptr=" <<*ptr;
  getch( );
  }
  *ptr=24
```

یہ میموری میں یوں ہوگا۔

ہم ایک بات اور بتائیں جو آپ کو پروگرام لکھتے وقت ذہن میں رکھنا ہوگی کہ & آپریٹر اور * آپریٹر ایک دوسرے کے الٹ ہیں۔ یعنی x==p اور p=&n لکھا جاتا ہے۔ آیئے اس کو ایک مثال سے سمجھتے ہیں۔

مثال نمبر 4.6 ایڈریس آپریٹر اور پوائنٹر آپریٹر

```
void main(void)
  {
  int temp=16;
  int* ptr=& temp;
```

```
int& refer=*ptr;
cout <<"ptr=" <<ptr <<endl;
cout <<"refer=" <<refer <<endl;
cout <<"*ptr=" <<*ptr;
getch( );
}
```

اس مثال میں آپ کو یہ بات سمجھانے کی کوشش کی گئی ہے کہ ریفرنس اور پوائنٹر آپریٹر کو کس طرح اکٹھا استعمال کرنا ہے۔اس پروگرام کی آؤٹ پٹ یہ ہوگی۔

```
ptr = 0x0fff4d1
refer = 16
* ptr = 16
```

پوائنٹر اینڈ اریز:

آپ پوائنٹرز پر حسابی آپریٹرز بھی لاگو کر سکتے ہیں۔اس کے علاوہ پوائنٹرز میں اضافہ اور کمی بھی کی جا سکتی ہے۔آپ سوچ رہے ہوں گے کہ پوائنٹر میں میموری کا ایڈریس محفوظ ہوتا ہے پھر یہ سب کیسے ہو سکتا ہے؟ تو آیئے اس کو ایک مثال سے دیکھتے ہیں۔

مثال نمبر 4.7 پوائنٹر ارے کا استعمال

```
void main(void)
{
const int temp=3; int answer=0;
int array[size]={20,29,36};
cout <<"size of(int)=" <<size of(int) <<endl;
int* team=array*temp-10;
for(int* i=array; i<temp; i++)
{
answer+=*i;
cout <<"\n i=" <<i;
cout <<"\n *i=" <<*i;
cout <<"\n Answer=" <<answer;
}
cout <<"\n team=" <<team;
getch( );
}
```

اس پروگرام کو ایگزیکیوٹ کرنے کے بعد آپ کو یہ آؤٹ پٹ حاصل ہوگی۔

```
size of(int)=2
i=0x8fb5ffee
*i=20
answer=20
i=0x8fbsfff0
*i=29
answer=49
team=0x8fb5fff2
```

ریفرنس ریٹرن کرنا:

آپ نے فنکشن کے بارے میں تفصیل سے پڑھا ہے اور اس بات سے بھی واقف ہیں کہ فنکشن کوئی نہ کوئی ویلیو ریٹرن بھی کر سکتے ہیں۔ اسی طرح فنکشنز کی مدد سے آپ ریفرنس بھی ریٹرن کر سکتے ہیں۔ یہ ویلیو پروگرامنگ زبان میں لوویلیو (Lvalue) کہلاتی ہے اور ایسی ویلیو کے لئے فنکشن کے لئے لوکل نہیں ہوتی۔

آیئے ایک ایسی مثال لکھتے ہیں جو ریفرنس ریٹرن کرتی ہو۔

مثال نمبر 4.8

```
int action(int& a, int& b)
    {
    if(a>b)
    return a;
    else return b;
    }
main(void)
    {
    int temp, temp1;
    cout <<"Enter two values:";
    cin >>temp >>temp1;
    cout <<temp <<"," <<temp1 <<"," <<action(temp, temp1);
    action(temp, temp1)=34
    cout <<endl;
    cout <<temp <<"," <<temp1 <<"," <<action(temp, temp1);
    getch( );
```

```
}
```

اس پروگرام کو ایگزیکیوٹ کرنے کے بعد اس کی آؤٹ پٹ کچھ یوں ہوگی۔

```
Enter two values:  17  21
17 , 21 , 21
17 , 34 , 34
```

آپ نے اس سے پہلے سٹرنگز کے سٹینڈرڈ فنکشنز پڑھے ہیں۔ اس میں ہم نے ایک پروگرام لکھا تھا جو ( )strchr اور ( )strstr فنکشنز استعمال کرتا تھا یہ مثال نمبر 3.14 تھی۔ اس میں آپ کو ٹائپ کاسٹنگ کی ضرورت پیش آئی جو پروگرام کوڈنگ کو کافی مشکل بنا رہی تھی۔ اس پروگرام کو آپ پوائنٹر کی مدد سے بھی حل کر سکتے ہیں۔ وہ کس طرح آیئے اس کے لئے ایک پروگرام لکھتے ہیں۔

مثال نمبر 4.9 strstr اور strchr فنکشن کا استعمال

```
#include<string.h>
#include<conio.h>
#include<iostream.h>
void main(void)
    {
    char str1[]="Failure is another stepping stone to greatness";
    cout <<"string=\" "<<str1 <<"\"\"n";
    char* temp=strchr(str1, 'l');
    cout <<"\n strchr(str1, 'l') located at=" <<temp-str1;
    temp=strchr(str1,'y');
    cout <<"\n strchr(str1,'g') located at=" <<temp-str1;
    temp=strstr(str1, "to");
    cout <<"\n strchr(str1,'to') located at=" <<temp-str1;
    temp=strstr(str1, "for");
    if(temp==Null)
    cout <<"\n Sorry Returns Null";
    getch( );
    }
```

اس پروگرام کی آؤٹ پٹ یہ ہے۔

```
string="Failure is another stepping stone to greatness"
strchr(str1, 'l') located at  = 3
```

```
strchr(str1, 'g') located at  = 26
strchr(str1, 'to') located at = 29
Sorry Returns Null
```

اس پروگرام میں ('l' ,strchr(temp کو سب سے پہلے کال کیا گیا ہے یہ سٹرنگ str1 میں موجود پہلے l کو ایک پوائنٹر ریٹرن کرتا ہے اور ایکسپریشن temp-str1 سٹرنگ میں اس کریکٹر کا انڈیکس نمبر معلوم کرتی ہے۔ آپ جانتے ہیں کہ ارے صفر(0) انڈیکس سے سٹارٹ ہوتی ہے۔
اس کے بعد ('g' ,strchr(temp کو کال کیا گیا ہے یہ سٹرنگ میں موجود سب سے آخری g کو پوائنٹر ریٹرن کرے گا اور اس کا انڈیکس نمبر ریٹرن کرتا ہے۔
اس کے بعد ہم نے سٹرنگ کے لئے فنکشن کال کیا ہے یعنی ( )strchr یہ کریکٹر کا انڈیکس نمبر معلوم کرنے کے لئے استعمال ہوتا ہے۔ آپ اس کے علاوہ سٹرنگ میں سے ایک سٹرنگ کا انڈیکس نمبر بھی معلوم کر سکتے ہیں۔ اس کے لئے ++C نے ایک ( )strstr کا فنکشن استعمال کرنے کی سہولت دی ہے۔ اس لئے فنکشن ("strstr(temp, "to کو کال سٹرنگ میں سے to کا انڈیکس نمبر ریٹرن کرے گا۔ یہ انڈیکس نمبر to سٹرنگ میں سے t کا ہوگا۔ یعنی جس کریکٹر سے سٹرنگ سٹارٹ ہوتی ہے اس کا یہ انڈیکس نمبر ہوگا۔
اس کے بعد ہم نے ("strstr(temp, "for کو کال کیا ہے۔ یہ سٹرنگ میں سے for کو پوائنٹر ریٹرن کرے گا اور اس کا انڈیکس نمبر ڈھونڈے گا۔ اس کے ساتھ ہم نے if کی کنڈیشن بھی لگائی ہے کہ اگر یہ سٹرنگ میں سے نہیں ملتا تو Null ریٹرن کرے گا اور وہ پرنٹ ہوگا۔

فنکشن پوائنٹر :

فنکشن کے پوائنٹر سے مراد ہے کہ اس کی ڈیفینیشن میں پوائنٹر استعمال کرنا۔ یہ پوائنٹر میموری میں فنکشن کا ایڈریس اپنے پاس محفوظ رکھتا ہے اور فنکشن پوائنٹر کسی بھی فنکشن کو پاس کیا جا سکتا ہے۔ یہ فنکشنز میں سے ریٹرن کروایا جا سکتا ہے۔ آپ اسے اریز میں سٹور کروا سکتے ہیں اور اس کے علاوہ دوسرے فنکشنز کے پوائنٹرز کو بھی آسائن کر سکتے ہیں۔ آپ نے پیچھے ارے کو ترتیب دینے کا پروگرام بنایا تھا۔ آیئے اب پوائنٹر فنکشن کی مدد سے یہ پروگرام حل کرتے ہیں۔

مثال نمبر 4.10 فنکشن پوائنٹر

```
#include<iomanip.h>
#include<conio.h>
#include<iostream.h>
void sorting(int[], const int, int(*) (int, int));
int ascend(int, int);
int descend(int, int);
void main(void)
    {
    int temp, count;
    int array[ ];
    cout <<"Enter any 12 Numbers" <<endl;
    for(int i=0; i<12; i++)
    cin >>array[i];
```

```
cout <<"Enter 2 to sort in ascending order:";
cout <<"\n Enter 1 to sort in descending order:";
cin >>temp;
cout <<"here is orignal array: \n";
for(count=0; count<12; count++)
cout <<array[count] <<setw(5);
if(temp==1)
{
sorting(array ,12, descend);
cout <<"\n Array elements in Descending order. \n";
}
else
{
sorting(array ,12, ascend);
cout <<"\n Array elements in Ascending order. \n";
}
for (counter=0; counter<12; counter++)
cout <<array[counter] <<setw(5);
getch( );
{
void sorting(int 1, cuonst int s, int(*match)(int ,int))
{
void arrange(int*, int*);
for(int i=1; i<s; i++)
for(int j=0; j<s; j++)
if((*match)(q[j], q[j+1]))
arrange(& q[i], & q[j+1]);
}
void arrange(int* data1, int* data2)
{
int temp;
temp=*data1;
*data1=*data2;
```

```
*data2=temp;
}
int ascend(int x, int y)
{
return y<x;
}
int descend(int a, int b)
{
return b>a;
}
```

اس پروگرام کو ایگزیکیوٹ کریں گے تو یہ آؤٹ پٹ سکرین پر ڈسپلے ہوگی۔

```
Enter any 12 Numbers
1   19  2   12  6   8   45  96  20  16  82  12  90
Enter 2 to sort in Ascending order:
Enter 1 to sort in Descending order: 1
Here is original array
1   19  2   12  6   8   45  96  20  16  82  12  90
Array elements is Descending order
96  90  82  45  20  19  16  12  8   6   2   1
```

اب آپ دوسری دفعہ پروگرام ایگزیکیوٹ کرتے ہیں اور اس دفعہ آپ آؤٹ پٹ Ascending آرڈر میں دیکھنا چاہتے ہیں تو 2 تحریر کیجئے گا۔ مثلاً

```
Enter any 12 Numbers
0   99  1   82  64  56  12  28  6   72  8   81
Enter 2 to sort in Ascending order:
Enter 1 to sort in Descending order: 2
Here is original array
0   99  1   82  64  56  12  28  6   72  8   81
Array elements is Ascending order
0   1   6   8   12  28  56  64  72  81  82  99
```

کریکٹر اور سٹرنگ فنکشنز:

آپ نے پہلے باب میں بھی سٹرنگ ہیڈر فائل میں موجود فنکشنز کے بارے میں پڑھا تھا۔ یہاں ہم پوائنٹر کی مدد سے آپ کو ان فنکشنز کے بارے میں بتائیں گے۔ نیچے ان کا ٹیبل درج ہے۔

فنکشن پروٹوٹائپ	فنکشن وضاحت
char *strcpy(char *s1, char *s2)	یہ سٹرنگ s2 کو s1 میں سٹور کرتا ہے اور s1 کی ویلیو ریٹرن کرتا ہے
char *strncpy(char *c, char *c1, 3)	یہ سٹرنگ c2 کے تین لفظ c1 ارے میں کاپی کرتا ہے
char *strcat(char *c, char *c1)	یہ سٹرنگ c2 کو c1 کے آخر پر لکھے گا c1 ارے کا آخری کریکٹر (Null) c2 کے پہلے کریکٹر سے تبدیل ہو جائے گا
char *strncat(char *c, char *c1, 5)	یہ سٹرنگ c2 کے 5 کریکٹرز (لفظ) c1 ارے میں لکھے گا
int strcmp(char *c, char *c1)	یہ سٹرنگ c1 کا c2 سے موازنہ کرتا ہے کہ کیا دونوں سٹرنگز برابر ہیں یا نہیں اگر برابر ہوں گے تو 1 ورنہ 0 ریٹرن کرتا ہے

یہ سٹرنگز کے کچھ اہم فنکشنز کی تفصیل تھی۔ اس سے پہلے آپ اس باب میں بھی سٹرنگز کے دو فنکشنز کو پوائنٹر کی مدد سے استعمال کرنے کا عمل پڑھ چکے ہیں۔ آیئے اب ایک اور پروگرام لکھتے ہیں۔

مثال نمبر 4.11 سٹرنگز اور فنکشنز کا استعمال

```
void main(void)
    {
    clrscr( );
    char *str1 = "Hello Sikandar";
    char *str2 = "How are you";
    char *str3 = "Hello Sikandar";
    cout <<"str1=" <<str1 <<"\n"
         <<"str2=" <<str2 <<"\n"
         <<"str3=" <<str3 <<"\n";
    cout <<"strcpy(str1, str2)=" <<strcpy(str1, str2);
    cout <<"\n strncpy (str2, str3, 4)=" <<strncpy (str2, str3, 4);
    cout <<"\n";
    cout <<"Length: \n strlen(str1)=" <<strlen(str1);
    getch( );
    }
```

آپ نے دیکھا کہ ہم نے پروگرام میں تین کریکٹر پوائنٹر انیشیلائیز کئے ہیں اور بعد میں ان کا موازنہ کیا ہے۔ اور ان میں سے str1 کی لمبائی معلوم کی ہے۔ جب آپ اس پروگرام کو ایگزیکیوٹ کریں گے تو یہ پروگرام یہ آؤٹ پٹ ڈسپلے کرے گا۔

```
str1 = Hello Sikandar
str2 = How are you
```

```
str3 = Hello Sikandar
strcpy(str1, str2) = How are you
strncpy(str2, str3, 6) = Hello e you
Length:
   strlen(str1) = 11
```

نیوآپریٹر (New):

آپ نے پوائنٹر ڈیکلیر کرنا سیکھا ہے اور اس کے استعمال سے بھی بخوبی واقف ہوں گے۔ اب ہم آپ کو تھوڑا سا ایڈوانس لے کر چلتے ہیں۔ آپ پوائنٹر ایسے ڈیکلیر کرتے ہیں۔

```
int *ptr;
```

اس کا مطلب ہے کہ ptr ویری ایبل int کے لئے ایک پوائنٹر ہے اسے انگلش میں یوں لکھتے ہیں۔

```
ptr is a pointer to an integer
```

یہ صرف پوائنٹر کے لئے میموری متعین کرتا ہے اور پوائنٹر کی ویلیو کوئی بھی میموری ایڈریس ہوگی لیکن دیکھیں کہ اس ایڈریس پر ابھی تک کوئی میموری متعین نہیں کی گئی۔ یعنی ptr اپنی شیلائیز نہیں کیا گیا اور یوں یہ کسی بھی متعین کی گئی ویلیو کو پوائنٹ بھی نہیں کر رہا۔ اب اگر آپ یوں لکھتے ہیں۔

```
*ptr = 81;       //Error message
```

تو یہ ایرر ہوگا یعنی یہ جس میموری کو پوائنٹ کر رہا ہے اگر آپ اسے ایکسیس کرنے کی کوشش کریں گے تو یہ ایک ایرر ہوگا کہ پوائنٹر ptr کے لئے کسی قسم کا ڈیٹا سٹور نہیں کیا گیا۔ ہم اس باب کے آغاز میں ایک بات واضح کر چکے ہیں کہ جب پوائنٹر ڈیکلیر کیا جاتا ہے تو یہ اسی وقت اپنی شیلائیز بھی کر دیا جاتا ہے یعنی اس ایرر سے بچنے کا بہترین طریقہ یہ ہے کہ آپ اسے اسی وقت اپنی شیلائیز کرا دیں جب پوائنٹر ڈیکلیر کیا جائے۔

```
int* temp = 171;            //temp = 171
int* ptr = & temp           //ptr = address of temp
```

اب اگر آپ ptr* کو ایکسیس کرتے ہیں۔

```
cout <<*ptr;
```

تو اس میں کوئی پرابلم نہیں ہے کیونکہ temp کا میموری ایڈریس ptr میں سٹور کیا جا چکا ہے۔ اس کے علاوہ اس کا ایک اور طریقہ بھی ہے جس کی مدد سے آپ اوپر والا کام بآسانی کر سکتے ہیں۔

```
int* prt;
ptr = new int;
*ptr = 171;
```

یہ پہلی دو لائنز آپ کمبائن بھی لکھ سکتے ہیں۔

```
int* ptr = new int;
```

new آپریٹر کا یہ فنکشن ہے کہ یہ میموری میں ایسی بائٹس کا ایڈریس ریٹرن کرتا ہے جو کسی اور ویری ایبل کے لئے ابھی تک ریزرو نہ کی گئی ہوں۔ اب جب آپ نے ptr=new int لکھا ہے تو اس کا یہ مطلب ہے کہ int کے لئے دو بائٹس ریزرو کر دے یعنی اسے بائیٹس کا ایڈریس دے دیا گیا ہے اور یہ اس بات کی وضاحت بھی کرتا ہے کہ prt اس وقت کسی اور ویری ایبل کے استعمال میں نہیں ہے۔ ایک اور بات کہ جب آپ new آپریٹر کے

ذریعے کسی پوائنٹر وریی ایبل کو انی شیلائیز کرتے ہیں تو وہ پوائنٹر انی شیلائیز ہوگا نہ کہ وہ میموری جس کو وہ پوائنٹ کر رہا ہوگا۔ ایسا ایک صورت میں ممکن ہے اگر آپ تمام کام ایک ہی لائن میں مکمل کریں۔ مثلاً

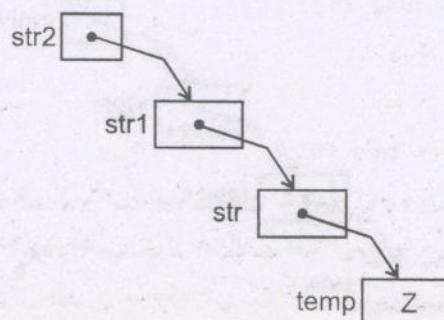
```
int* ptr = new int(171);   //both and * ptr have been initialized
```

ڈیلیٹ (Delete) آپریٹر:

یہ آپریٹر new آپریٹر کا متضاد ہے یعنی یہ اس کا عمل تبدیل کر دیتا ہے مثلاً آپ new کی مدد سے کسی پوائنٹر کے لئے میموری متعین کرتے ہیں اور ڈیلیٹ کی مدد سے آپ اس میموری کو پھر سے آزاد یعنی خالی کروا سکتے ہیں۔ اس سے یہ واضح ہوا کہ delete آپریٹر میموری کو خالی کروانے کے لئے استعمال ہوتا ہے۔ یہ کس طرح استعمال کیا جانا چاہئے آیئے ایک نظر دیکھتے ہیں۔

```
float* ptr = new float(7.219);
delete ptr;
cout <<*ptr;          //Error ptr has been deallocated
```

جب ایک دفعہ آپ کسی بھی پوائنٹر وریی ایبل کو deallocate کر دیتے ہیں تو یہ اس وقت تک دوبارہ استعمال نہیں کیا جا سکتا جب تک کہ reallocate نہ کیا جائے۔ ایک deallocate کیا ہوا پوائنٹر dangling پوائنٹر بھی کہلاتا ہے۔

ایک بات اور جو قابلِ غور ہے کہ delete آپریٹر صرف اس وقت استعمال کیا جا سکتا ہے جب آپ کسی پوائنٹر کو new آپریٹر کی مدد سے انی شیلائیز کرتے ہیں اور آپ کانسٹنٹ پوائنٹر کو deallocate نہیں کر سکتے۔ مثلاً

```
const int* ptr = new int;
delete ptr;     //Error cannot delete pointer to const
```

پوائنٹرز ٹو پوائنٹر:

اس سے پہلے تک ہمارا ہر پوائنٹر میموری ایڈریس یا ویلیو کو پوائنٹ کرتا تھا لیکن ایک پوائنٹر کسی دوسرے پوائنٹر کو بھی پوائنٹ کر سکتا ہے۔ مثلاً

```
char temp = 'Z';
char* str = & temp;
char** str1 = & str;
char*** str2 = & str1;
***str2 = 'S';
```

یہ میموری میں کچھ یوں سٹور ہوں گے۔

اریز:

آپ اریز سے بخوبی واقف ہیں لیکن یہاں پر ہم آپ کو اریز کے بارے میں مزید کچھ بتائیں گے۔ آپ نے اب تک ایسی اریز بنائی ہیں جو کہ کمپائل ٹائم پر بنتی ہیں اور اس کی میموری اسی وقت ویری ایبلز کی لئے متعین کر دی جاتی ہے۔ ایسی اریز کو static ارے کہتے ہیں۔ آپ اس کے علاوہ ایسی اریز بھی بنا سکتے ہیں جو رن ٹائم پر بنتی ہیں اور اس کی میموری ویری ایبلز کے لئے اس وقت متعین کی جاتی ہے جب اس ارے کی ڈیکلیریشن ایگزیکیوٹ ہو۔ ایسی اریز کس طرح بنائی جا سکتی ہیں اور یہ کس طرح کام کرتی ہیں۔ آیئے اس کی ایک مثال دیکھتے ہیں۔

مثال نمبر 4.12 dynamic ارے

```
void input(double*&, int&);
void output(double*, int);
void main(void)     {
    clrscr( );
    double* temp;
    int q;
    input(temp, q);
    output(temp, q);
    delete[ ] temp;
    input(temp, q);
    output(temp, q);
    delete[ ] temp;
    getch( );
    }
void input(double*& temp, int& q)
    {
    cout <<"\n How many values you want to Enter:";
    cin >>q;
    temp=new double[q];
    cout <<"\n Enter values one per one line: \n";
    for(int i=0; i<q; i++)
    {
    cout <<i+1 <<":";
    cin >>temp[i];
    }
    }
```

```
void output(double* temp, int q)
    {
    for(int i=0; i<q; i++)
    cout <<temp[i] <<" ";
    }
```

اس پروگرام میں ہم نے دو فنکشنز بنائے ہیں۔ایک یوزر سے اریز کی ویلیوز لیتا ہے جبکہ دوسرا وہی ویلیوز پرنٹ کرواتا ہے۔اس پروگرام میں ہم ایسی ارے بنا رہے ہیں جو رن ٹائم پر ایگزیکیوٹ ہوگی اور یوزر سے پوچھے گی کہ وہ کتنی ویلیوز ارے میں درج کرنا چاہتا ہے۔آیئے اس پروگرام کی آؤٹ پٹ دیکھتے ہیں۔

```
How many values you want to Enter:    4
Enter one value per one line
1   :   12
2   :   36
3   :   91
4   :   56
12      36      91      56
How many values you want to Enter:    2
Enter one value per one line
1   :   102
2   :   371
102     371
```

سوال نمبر 1: پوائنٹر زکیا ہیں اور یہ کس لئے استعمال کئے جاتے ہیں؟

سوال نمبر 2: ایک ایسا پروگرام بنائیں جس میں ایک فنکشن کوارے پاس کی گئی ہو؟ اس کے لئے آپ کو پوائنٹر استعمال کرنا ہوگا۔

سوال نمبر 3: ایک ایسا پروگرام بنائیں جو یوزر سے دوسٹرنگز لے اور ان کا موازنہ کرے کہ کیا وہ برابر ہیں یا نہیں۔ اس میں آپ نے <string.h> فنکشن استعمال نہیں کرنا۔ یعنی strcmp فنکشن استعمال نہیں کرنا بلکہ پوائنٹر اور یوزر ڈیفائن فنکشن استعمال کرنا ہے۔

سوال نمبر 4: آپ نے مشہور سیریز f(0)+f(1)+f(2)+.........f(n–1) کے بارے میں پڑھا ہوگا۔ اس کو کیلکولیٹ کرنے کے لئے ایک پروگرام تحریر کریں جس میں پوائنٹر کا استعمال ہو اور جو فنکشن بنائیں اس کو پوائنٹر پاس کیا گیا ہو۔ مثلاً یہ سیریز یوں عمل کرتی ہے۔

1 + 3 + 6 + 11 = 21

سوال نمبر 5: ڈیلیٹ اور new آپریٹر میں کیا فرق ہے؟

سوال نمبر 6: پروگرامنگ کے ان کوڈز میں کیا ایرر ہے؟

```
(i)    int* compare;
       int* temp = &compare;
(ii)   char a = 'z';
       char b = &a';
(iii)  int i = new int;
       int* j = new int
(iv)   int b[20];
       for(int i=0; i<20; i++)
       *b++ = i*i;
```

جوابات

1: جواب

ایک وری ایبل اپنی ویلیو کو ڈائریکٹ ریفرنس کرتا ہے جبکہ ایک پوائنٹر اپنی ویلیو کو ان ڈائریکٹ ریفرنس کرتا ہے یعنی وری ایبل میں اس کی ویلیو سٹور ہوتی ہے جبکہ پوائنٹر میں ایک وری ایبل کا میموری ایڈریس ہوتا ہے اور یہ ایڈریس آپ کی اس ویلیو کی نشاندہی کرتا ہے۔ دوسرے وری ایبلز کی طرح پوائنٹرز بھی استعمال کرنے سے پہلے ڈیکلیر کرنا ضروری ہوتے ہیں۔

2: جواب

```
#include<conio.h>
#include<iostream.h>
#include<iomanip.h>
void resultant(double*);
const int size=6;
main(void)
    {
    clrscr( );
    double array[size] = {12,15,91.5,56,32.6,78};
    cout <<"Your origional array" <<endl;
    for(int i=0; i<size; i++)
    cout <<array[i] <<",";
    cout <<endl;
    resultant(array);
    for(int i=0; i<size; i++)
    cout <<"Array[" <<i <<"]=" <<array[i] <<endl;
    getch( );
    }
 void resultant(double* temp)
    {
    for(int i=0; i<size; i++)
    *temp++*=3;
    }
```

3: جواب

```
#include<conio.h>
#include<iostream.h>
#include<stdio.h>
int relastic(const char*, const char*);
main(void)
    {
    clrscr( );
    int compare;
    char str1[75], str2[75];
    cout <<"Enter 1st String:";
    gets(str1)
    cout <<"\n Enter 2nd String:";
    gets(str2)
    compare = relastic(str1, str2);
    if(compare==1)
    cout <<str1 <<"!=" <<str2;
    else
    cout <<str1 <<"==" <<str2);
    getch( );
    }
int relastic(const char *n1, const char *n2)
    {
    //while(*n1 ! = '\o')
    //++n1;
    for(; *n1 ! = '\o' && *n2 ! = 'o'; n1++, n2++)
    if(*n1 ! = *n2)
    return 0;
    return 1;
    }
```

❹: جواب

f(0)+f(1)+f(2)+...........f(n) معلوم کرنے کے لئے یہ پروگرام بنائیں۔

```
int result(int(*)(int), int);
int dem(int);
void main( )
   {
   clrscr( );
   int val;
   cout <<"Enter Number of values to be Calculated:";
   cin >>val;
   cout <<result(dem, val);
   getch( );
   }
int result(int(*temp)(int a), int b)
   {
   int ans=0;
   for(int i=0; i<=b; i++)
   ans+=(*temp)(i);
   return ans;
   }
int dem(int a)
   {
   return a×d;
   }
```

❺: جواب

new کی ورڈ کسی بھی پوائنٹر کے لئے میموری ریزرو کرتا ہے۔ یہ میموری میں ایسی بائٹس کا ایڈریس ریٹرن کرتا ہے جو کسی اور ویری ایبل کے لئے ابھی تک ریزرو نہ کی گئی ہوں۔

delete کی ورڈ new آپریٹر کے الٹ ہے۔ یہ new کی مدد سے پوائنٹر کے لئے ریزرو کی ہوئی میموری کو ختم کرتا ہے۔ یعنی delete کی مدد سے آپ new کی مدد سے ریزرو کی ہوئی میموری کو پھر سے آزاد (خالی) کروا سکتے ہیں۔

❻: جواب

(i) **Error**

```
cannot convert 'int**' to 'int*'
```

(ii) **Error**

```
cannot convert char* to char
```

(iii) **Error**

```
cannot convert int* to int
```

(iv) **Error**

```
value required
```

باب نمبر 5

سٹرکچر اینڈ کلاسز

ایک سٹرکچر سادہ ویری ایبلز کا مجموعہ ہوتا ہے اور یہ ویری ایبلز کسی بھی ٹائپ کے ہو سکتے ہیں اور سٹرکچر میں شامل اجزاء سٹرکچر کے ڈیٹا ممبر کہلاتے ہیں۔ درحقیقت سٹرکچرز بنانے کا طریقہ تقریباً کلاسز سے ملتا ہے۔ فرق صرف یہ ہے کہ سٹرکچر ڈیٹا کا مجموعہ ہوتا ہے جبکہ کلاسز میں ڈیٹا اور فنکشنز دونوں شامل ہوتے ہیں اور کلاسز کے ان فنکشنز کو ممبر فنکشن کہتے ہیں۔ آپ فنکشنز کے بارے میں تفصیل سے پڑھ چکے ہیں اور اس باب کے پہلے حصہ میں آپ سٹرکچر کے بارے میں پڑھیں گے اور بعد میں آپ کلاسز کے بارے میں معلومات حاصل کریں گے۔

اس سے پہلے آپ نے جو کچھ پڑھا وہ اوبجیکٹ اوورینٹڈ ڈیزائن (OOD) کہلاتا ہے اور اس باب میں اوبجیکٹ اوورینٹڈ پروگرامنگ (OOP) کے بارے میں پڑھیں گے۔ یہ C++ میں پروگرامنگ کا یونٹ کلاس ہوتا ہے جس کا بعد میں اوبجیکٹ بنایا جاتا ہے۔ اس باب میں آپ ممبر ڈیٹا اور ممبر فنکشن دونوں کے بارے میں تفصیل سے پڑھیں گے۔

- سٹرکچر
- سٹرکچر اجزاء کو ایکسیس کرنا
- سٹرکچر پوائنٹر
- Nested سٹرکچرز
- یوزر ڈیفائنڈ ڈیٹا ٹائپس
- کلاسز
- کلاس ڈیکلیریشن
- پرائیویٹ اور پبلک کی ورڈز
- کنسٹرکٹر
- کنسٹرکٹر اوورلوڈنگ
- اوبجیکٹ بطور آرگومنٹس
- فنکشن سے اوبجیکٹ ریٹرن کرنا
- ڈیسٹرکٹر
- مشق

سٹرکچرز:

سٹرکچر ویری ایبلز کے مجموعہ کو کہتے ہیں اور سٹرکچر میں شامل ویری ایبلز کسی بھی ٹائپ کے ہو سکتے ہیں۔ مثلاً یہ ویری ایبلز int ٹائپ کے بھی ہو سکتے ہیں اور ان میں سے بعض double ٹائپ کے بھی ہو سکتے ہیں۔ سٹرکچر میں شامل ڈیٹا آئٹمز (ویری ایبلز وغیرہ) سٹرکچر کے ممبر (ارکان) کہلاتے ہیں۔ آپ کسی بھی قسم کا سٹرکچر یوں بناتے ہیں۔

```
struct employee
{
int emp;
float temp;
char choice;
};
```

جب بھی آپ سٹرکچر بنانا چاہتے ہیں تو اس کے لئے کی ورڈ struct استعمال کریں گے۔ employee ایڈنٹیفائر سٹرکچر ٹیگ ہے یا یہ سٹرکچر کا نام ہے اور یہ سٹرکچر ٹائپ کے ویری ایبلز کو ڈیکلیر کرنے کے لئے استعمال ہوتا ہے۔ سٹرکچر بریکٹس ({ }) کے اندر جو نام ڈیکلیر کئے گئے ہیں وہ سٹرکچر ممبرز ہیں۔ ایک اہم بات یہ ہے کہ سٹرکچر کی ڈیفینیشن میں جو ویری ایبلز ڈیکلیر کئے جائیں گے ان کے نام منفرد ہونا ضروری ہیں اور سٹرکچر کا اختتام سیمی کالن (;) سے ہونا ضروری ہے۔ ایک سٹرکچر خود اپنا انسٹینس (instance) نہیں بنا سکتا۔ اب یہ انسٹینس کیا ہے آپ آگے تفصیل سے پڑھیں گے۔ فی الحال آپ صرف یہ خیال رکھیں کہ آپ سٹرکچر کے نام کا ایڈنٹیفائر سٹرکچر کی باڈی میں ڈیکلیر نہیں کر سکتے۔ یعنی employee سٹرکچر ممبر کا نام employee نہیں ہو سکتا۔

سٹرکچر اجزاء (اراکین، ممبر یا ویری ایبلز) کو ایکسیس کرنا:

آپ نے اوپر سٹرکچر کے ممبرز ڈیکلیر کئے ہیں اب آپ اگر ان کو سٹرکچر کی باڈی سے باہر ایکسیس کرنا چاہتے ہیں اس کے لئے آپ کو ممبر ایکسیس آپریٹر استعمال کرنا ہو گا۔ سٹرکچر کے لئے ممبرز کو ایکسیس کرنے کے لئے دو ایکسیس آپریٹرز استعمال کئے جاتے ہیں۔ ڈاٹ آپریٹر (.) اور ایرو آپریٹر (→)

ڈاٹ آپریٹر کی مدد سے آپ سٹرکچر کے ممبرز (ویری ایبلز) کو ایکسیس کرتے ہیں۔ یعنی اس کی مدد سے آپ اوبجیکٹ کا ویری ایبل نام یا اوبجیکٹ کا ریفرنس ایکسیس کرتے ہیں۔ مثلاً ہم نے employee کا اوبجیکٹ emple بنایا ہے اور آپ temp ویری ایبل کو ایکسیس کرنا چاہتے ہیں تو اسے یوں ایکسیس کر سکتے ہیں۔

```
cout <<emple.temp;
```

ایرو آپریٹر آپ اس وقت استعمال کریں گے جب آپ نے سٹرکچر ممبر کو ایکسیس کرنے کے لئے اوبجیکٹ کو پوائنٹر ڈیکلیر کیا ہو۔ فرض کریں کہ empptr* پوائنٹر ڈیکلیر کیا گیا ہے جو employee اوبجیکٹ کو پوائنٹ کرتا ہے۔ تو جب آپ اس کی مدد سے سٹرکچر ممبر کو ایکسیس کرنا چاہیں گے تو ایرو آپریٹر استعمال کریں گے۔ ایک اور اہم بات یہ ہے کہ ایرو آپریٹر لکھا کس طرح جاتا ہے۔ اس کے لئے تفریقی علامت (minus operator –) اور اس کے بعد (>) greater علامت لکھیں اور ان کے درمیان سپیس نہیں ہونی چاہئے۔ یہ بھی ہو سکتا ہے کہ یہ آپ کے لئے کنفیوزنگ پوائنٹ ہو لیکن جب آپ اس کی مثال دیکھیں گے تو یہ کلیئر ہو جائے گا۔

سٹرکچر کس طرح بنایا جاتا ہے اور کس طرح اس کے ڈیٹا ممبر کو ایکسیس کیا جا سکتا ہے آیئے اس کی ایک مثال دیکھتے ہیں۔

مثال نمبر 5.1 سٹرکچر بنانا

```
struct student
    {
    char name[20], & course[5];
    int rollno;
    };
void main(void)
    {
    clrscr( );
    student st1, st2;       //variable of new type student
    cout <<"\n Enter Student1 Name:";
    cin >>st1.name;
    cout <<"\n Enter Student1 class:";
    cin >>st1.course;
    cout <<"\n Enter Student1 rollno:";
    cin >>st1.rollno;
    cout <<"\n Enter Student2 Name:";
    cin >>st2.name;
    cout <<"\n Enter Student2 class:";
    cin >>st2.course;
    cout <<"\n Enter Student2 rollno:";
    cin >>st2.rollno;
    getch( );
    }
```

اس پروگرام میں ہم نے ایک student کے نام سے سٹرکچر بنایا ہے اور اس میں تین ویری ایبلز ڈیکلیر کئے ہیں۔ اس کے بعد ہم نے main( ) میں سٹرکچر ٹائپ کے دو ویری ایبلز ڈیکلیر کئے ہیں۔

```
student st1, st2;
```

آپ ان کو سٹرکچر کے اوبجیکٹ بھی کہہ سکتے ہیں۔ ان کی مدد سے ہم سٹرکچر کے ڈیٹا ممبرز ایکسیس کریں گے۔ اس لائن سے یہ بھی ظاہر ہوا کہ آپ ایک سٹرکچر کے کئی اوبجیکٹس بھی بنا سکتے ہیں یا سٹرکچر ٹائپ کے کئی ویری ایبلز بھی ڈیکلیر کر سکتے ہیں۔ بعد میں ہم نے student1 کے متعلق ڈیٹا یوزر سے حاصل کیا ہے اور اس کے لئے ہم نے ویری ایبل st1 استعمال کیا ہے جبکہ student2 کا ڈیٹا حاصل کرنے کے لئے st2 استعمال کیا ہے۔ یہ ایک بالکل سادہ سا پروگرام ہے جو صرف آپ کو یہ بات واضح کرتا ہے کہ کس طرح سٹرکچر بنتا ہے اور کیسے اس کو بعد میں استعمال کیا جا سکتا ہے اور یہ اوبجیکٹ

پروگرامنگ کی طرف پہلا قدم ہے کہ ہم نے ایک نام کے دو آوبجیکٹ بنائے ہیں اور پھر وہ آگے استعمال کئے ہیں۔ اس پروگرام میں ہم نے ہر سٹوڈنٹ کے لئے الگ سے ان پٹ لی ہے۔ یہاں پر صرف دو ریکارڈز کی ہمیں ضرورت تھی لیکن اگر آپ کو فرض کریں 25 ریکارڈز چاہئیں تو کیا ہر ایک کے لئے الگ سے ان پٹ لیں گے؟ تو ایسا نہیں ہے اس کے لئے ہم ارے کا استعمال کریں گے اور لوپ کی مدد سے ان پٹ لیں گے۔ آیئے اس کے لئے ایک سٹرکچر پروگرام لکھتے ہیں۔

مثال نمبر 5.2 ریکارڈ تلاش کرنا

```
struct student
    {
    char name[20], aclass[5], address[40];
    int rollno;
    };
void main(void)
    {
    clrscr( );
    student st[3];
    int count, tryno;
    for(count=1; count>=3; count++)
    {
    cout <<"\n Enter student" <<count <<"Rollno:";
    cin >>st[count].rollno;
    cout <<"\n Enter student" <<count <<"Name:";
    cin >>st[count].name;
    cout <<"\n Enter student" <<count <<"Class:";
    cin >>st[count].aclass;
    cout <<"\n Enter student" <<count <<"Address:";
    cin >>st[count].address;
    }
    cout <<"\n Enter Rollno to find student:";
    cin >>tryno;
    for(count=1; count<=3; count++)
    {
    if(tryno==st[count].rollno)
    {
```

```
cout <<"student rollno:" <<st[count].rollno;
cout <<"\n student name:" <<st[count].name;
cout <<"\n student class:" <<st[count].aclass;
cout <<"\n student address:" <<st[count].address;
}
}
getch( );
}
```

اس پروگرام میں ہم نے ایک سٹوڈنٹ نام کا سٹرکچر بنایا ہے اور اس میں چار ویری ایبلز ڈیکلیر کئے ہیں۔ ( )main میتھڈ میں سٹوڈنٹ کا اوبجیکٹ st بنایا ہے اور یہ ایک ارے ٹائپ کا اوبجیکٹ ہے۔ جس کو ویلیو پاس کی گئی ہے۔ for کے لوپ کی مدد سے ہم نے ہر سٹوڈنٹ کے لئے ان پٹ لی ہے یعنی اس کا یہ فائدہ ہے کہ ہم نے خواہ کتنے ریکارڈز یوزر سے کیوں نہ لئے ہوں ہم صرف ایک سٹوڈنٹ کی کوڈنگ کرتے ہیں اور for لوپ کی مدد سے اسے کنٹرول کرتے ہیں۔ اس کے بعد ہم نے یوزر سے ایک اور ان پٹ لی ہے۔

```
cin>> tryno;
```

اب یہاں یوزر جو بھی نمبر تحریر کرے گا اس کو اوپر لکھے گئے ریکارڈز میں موجود رول نمبر کے ساتھ ملایا جائے گا۔ اگر یہ نمبر مل گیا تو اس سے متعلقہ ڈیٹا شو کر دیا جائے گا۔ اب یہ نمبر سٹوڈنٹ کے رول نمبر کے برابر ہے یا نہیں؟ اس کے لئے if سٹیٹ مینٹ استعمال کی گئی ہے۔

```
if(tryno==st[count].rollno)
```

یہ سٹیٹ مینٹ tryno کو ہر رول نمبر کے ساتھ ملائے گی اور ریٹرن کرے گی کہ کیا یہ نمبر کسی ریکارڈ سے ملتا ہے یا نہیں۔

سٹرکچر پوائنٹر :

ہم نے اس باب کے آغاز میں آپ کو بتایا تھا کہ آپ ایک سٹرکچر کو پوائنٹر بھی پاس کر سکتے ہیں اور ایسے سٹرکچر کے ویری ایبلز یا ڈیٹا ممبرز کو ایکسیس کرنے کے لئے پوائنٹر آپریٹر استعمال کیا جاتا ہے۔ آپ یہ کام کس طرح پرفارم کر سکتے ہیں آیئے اس پروگرام کی مدد سے سمجھنے کی کوشش کرتے ہیں۔

مثال نمبر 5.3 سٹرکچر پوائنٹر

```
struct show
    {
    char name[20], sex[6];
    int age;
    };
show* inpt(void);
void output(show*);
void main(void)
    {
    clrscr( );
```

```
    show* sh;
    sh=input( )
    output(sh);
    getch( );
    }
show* input(void)
    {
    show* temp;
    cout <<"\n Enter your Age:";
    cin >>temp→age;
    cout <<"\n Enter your Name:";
    gets (temp→name);
    cout <<"\n Enter your Sex:";
    gets (temp→sex);
    return temp;
    }
void output(show* temp)
    {
    cout <<"\n Your Age:" <<temp→age;
    if(temp→age<=10)
    cout <<"\n You are a child" <<temp→sex;
    else
    if(temp→age<=20)
    cout <<"\n You are a teenage" <<temp→sex;
    else
    if(temp→age<=30)
    cout <<"\n You are an adult" <<temp→sex;
    else
    cout <<"\n You should now take rest: an old" <<temp→sex;
    }
```

آپ جب اس پروگرام کو رن کریں گے تو آؤٹ پٹ یہ ہوگی۔

```
Enter you Age: 22
```

```
Your Name:     Yasir Aurangzaib
Your Sex:      Male
Your Age 22
You are an adult male
```

اس پروگرام میں ہم نے ایک سٹرکچر show بنایا ہے اور دو فنکشنز لکھے ہیں۔ ان میں سے ایک فنکشن input( ) سٹرکچر پوائنٹر ویلیو ریٹرن کرتا ہے جبکہ دوسرا فنکشن output( ) کو سٹرکچر پوائنٹر بطور پیرامیٹر پاس کیا ہے۔ اب آپ سوچ رہے ہوں گے کہ پوائنٹر سٹرکچر کس طرح پاس کیا جاتا ہے تو اس کا حل main( ) میں موجود ہے۔ ہم نے سٹرکچر کا اوبجیکٹ sh بنایا ہے اور یہ پوائنٹر اوبجیکٹ ہے پھر اس کو ایک فنکشن میں سے پاس کر دیا ہے جبکہ دوسرے فنکشن کو اس اوبجیکٹ میں محفوظ کیا ہے۔

فنکشن input( ) میں ہم نے یوزر سے ان پٹ لی ہے اور سٹرکچر کے ویری ایبلز استعمال کئے ہیں۔ اب سٹرکچر کے ویری ایبلز استعمال کرنے کے لئے ان کو ایک ریفرنس کی ضرورت ہے اس کے لئے ہم نے اس فنکشن میں temp پوائنٹر اوبجیکٹ بنایا ہے اور یہ پوائنٹر ہے۔ اس لئے ویری ایبلز کو پوائنٹر ایکسیس آپریٹر (→) کی مدد سے ایکسیس کیا ہے اور آخر میں پوائنٹر ریٹرن کیا ہے یعنی پوائنٹر کا اوبجیکٹ temp ریٹرن کیا ہے۔

output( ) فنکشن میں ہم نے پوائنٹر اوبجیکٹ بطور آرگومنٹس پاس کیا ہے اور اس کو مزید پراسسنگ کے لئے استعمال کیا ہے۔

Nested سٹرکچرز:

آپ سٹرکچرز میں مزید سٹرکچر بھی لکھ سکتے ہیں اور یہ کس طرح کام کرے گا اور اس کو لکھنے کا کیا طریقہ ہے آیئے اس کے لئے ایک پروگرام لکھتے ہیں۔

<u>مثال نمبر 5.4</u>

```
struct premier
    {
    int pm;
    float cm;
    };
struct inner
    {
    premier height;
    premier width;
    };
void main(void)
    {
    clrscr( );
    inner inn;                        //instance of inner
    float temp1, temp2;
    inn.heiht.cm=5.16;                //assign values to inner
```

```
inn.width.pm=11;
cout <<"\n Enter length of ground in inches:";
cin >>inn.height.cm;
cout <<"\n Enter width of ground in feet:";
cin >>inn.height.pm;
temp1=inn.height.pm+inn.height.cm/12;
temp2=inn.width.pm+inn.width.cm/12;
cout <<"\n Ground area is:" <<setprecision(2) <<(temp1*temp2);
cout <<"squarefeet";
getch( );
}
```

اس پروگرام میں ہم نے دو سٹرکچرز بنائے ہیں اور ایک سٹرکچر کا انسٹینس ویری ایبل ڈیکلیر کیا ہے اور اس ایک ویری ایبل کو دو مختلف سٹرکچر کی ویلیو آسائن کی ہیں۔ اس پروگرام میں پہلے premier نام کا سٹرکچر بنایا ہے اور دوسرا سٹرکچر inner ہے۔ اس inner سٹرکچر میں دو ویری ایبلز ہیں جن کی ٹائپ premier (یعنی پہلا سٹرکچر) ہے۔ اس پروگرام کو جب آپ ایگزیکیوٹ کریں گے تو یہ چار ویلیوز کو کیلکولیٹ کرے گا کیونکہ آپ کے پاس چار ویری ایبلز ہیں دو پہلے سٹرکچر کے اور دو دوسرے سٹرکچر میں ڈیکلیر کئے گئے ہیں۔ ایک اہم بات کہ آپ main ( ) میں اس سٹرکچر کا اوبجیکٹ یا انسٹینس ڈیکلیر کریں گے۔ جس میں پہلے سٹرکچر ٹائپ کے ویری ایبلز ڈیکلیر ہوں گے۔ اس پروگرام کی آؤٹ پٹ یہ ہوگی۔

```
Enter length of ground in feet:   14
Enter length of ground in inches: 16.21
Ground area is:    178.22 square feet
```

یوزر ڈیفائنڈ ڈیٹا ٹائپس:

آپ نے C++ کی ڈیٹا ٹائپس (int, char) کے بارے میں پڑھا ہے۔ C++ ان ٹائپس کے علاوہ یہ بھی سہولت فراہم کرتی ہے کہ آپ اپنی خود سے سپیشل ڈیٹا ٹائپس بھی بنا سکتے ہیں۔ آپ ایسا کئی طریقوں سے کر سکتے ہیں۔ لیکن ہم یہاں پر ایک سادہ یوزر ڈیفائن ڈیٹا ٹائپ کے بارے میں بتائیں گے۔ آپ enumeration کی مدد سے خود ایک integer ڈیٹا ٹائپ بنا سکتے ہیں۔ اس کا جنرل طریقہ یہ ہے۔

```
enum typename{enumerator list};
```

یہاں پر سٹارٹ میں جو enum لکھا ہے C++ کا کی ورڈ ہے۔ typename سے مراد کوئی بھی ایڈنٹیفائر ہے جو یہ ظاہر کرتا ہے کہ یہ ابھی ڈیفائن کی گئی ٹائپ کا نام ہے اور لسٹ میں ڈیفائن کئے گئے کانسٹنٹ ایڈنٹیفائر کے نام ہوں گے۔ مثلاً

```
enum Sex(Male, Female};
```

اس سے آپ کا کوڈ آسان ہو جاتا ہے لیکن ایک بات کا خیال رکھیں کہ enum کا بہت زیادہ استعمال نہ کریں۔ وہ اس لئے کہ enumlist میں ہر ایڈنٹیفائر ایک کانسٹنٹ ہوتا ہے اور آپ اسے اپنے پروگرام میں مقصد کے علاوہ کہیں بھی استعمال نہیں کر سکتے۔ ایک بات اور کہ enum میں ڈیکلیر

کئے ہوئے ایڈنٹیفائر درست ہونے ضروری ہیں۔ اس طرح کے ایڈنٹیفائر آپ ڈیکلیر نہیں کر سکتے۔

```
enum Declare{c-, a+b, A, D};          //Error
```

آیئے اس کے لئے ایک سادہ پروگرام لکھتے ہیں تاکہ آپ کو اس کے استعمال کرنے کا بہتر طریقہ آ جائے۔

مثال نمبر 5.5

```
    //user defined data types
enum Days{mon, tues, thur, wed, fri, sat, sun};
void main(void)
    {
    clrscr( );
    Days day1, day2;
    day1 = wed;
    day2 = sun;
    int temp;
    if(day1<day2)
    {
    temp = day2-day1;
    cout <<"\n Day(s) Between=" <<temp;
    cout <<"\n Day1 comes before day2";
    }
    else
    {
    temp = day1-day2;
    cout <<"\n Days between=" <<temp;
    cout <<"\n day1 comes after day2";
    }
    getch( );
    }
```

اس پروگرام میں ہم نے enum ڈیکلیریشن میں وہ تمام نام لکھے ہیں جو کہ اس ٹائپ کی ویلیوز ہوں گی یعنی آپ اس ٹائپ کے ویری ایبلز کو یہ ویلیوز آسائن کریں گے۔ ان ویلیوز کو اینمیریٹر کہتے ہیں۔ جب آپ ایک دفعہ enum ٹائپ ڈیکلیر کر لیتے ہیں تو اس کے بعد آپ اس ٹائپ کے ویری ایبلز بنانے کے اہل ہو جاتے ہیں۔

```
Days day1, day2;
```

آیئے اس کے لئے ایک اور بولین ٹائپ کا پروگرام لکھتے ہیں۔

مثال نمبر 5.6 //word count

```
enum phrase{No, Yes};                //No=0, Yes=1
void main(void)
 {
  clrscr( );
  phrase ph;
  ph = No;
  char ch;
  int count = 0;
  cout <<"Enter a phrase:";
  do {
  ch = getche( );                    //read, get character
  if(ch==' '|| ch== '\r')            //if space
  {
  if(ph==Yes)                        //if word
  {
  count++;                           //count word
  ph = No;                           //for again count reset
  }
  }
  else                               //a normal word
  if(ph==No)                         //start of word
  ph=Yes;                            set flag
  }
  while(ch!= '\r');                  //end of Enter Key
  cout <<"\n word(s) count are:" <<count;
  getch( );
  }
```

جب آپ اس پروگرام کو ایگزیکیوٹ کریں گے تو آوٹ پٹ کچھ اس طرح ڈسپلے ہوگی۔

```
Enter a phrase:     I Love Islam
```

```
word(s) count are: 3
```

نوٹ: آپ یہ جملہ جتنا چاہیں مرضی لکھ لیں لیکن صرف ایک لائن پر لکھ سکتے ہیں۔ جونہی Enter پریس کریں گے آپ کا جملہ ختم ہوجائے گا۔

اس پروگرام میں ہم نے ایک phrase نام سے enum ڈیکلیر کی ہے جس کی دو ویلیوز ہیں۔ Yes اور No اس میں Yes ویلیو 1 اور No کی ویلیو 0 ہوگی۔ اور سٹارٹ میں ph ویری ایبل اینی شلائز کیا ہے۔ یہ phrase ٹائپ کا ویری ایبل ہے بعد میں یوزر سے do لوپ میں ان پٹ لی ہے۔ ;( )ch=getch یہ فنکشن اس وقت تک ان پٹ دے گا جب تک کہ آپ Enter پریس نہیں کرتے۔ اس کے بعد if کنڈیشن استعمال کی ہے کہ اگر لکھی گئی ان پٹ میں سپیس ہے اور کوئی جملہ بھی ہے تو ++count یعنی count ویری ایبل میں ایک کا اضافہ کردے اور ph=no کردے تاکہ کرسر اگلے لفظ کو پڑھ سکے اور اگر ایسا نہیں ہے تو جتنے لفظ لکھے گئے ہیں اس کی تعداد پرنٹ کردے۔

کلاسز: (Classes):

آپ نے ابھی تک ++C پروگرامنگ لینگوئج سے متعلق بنیادی پروگرامنگ پڑھی ہے لیکن اب آپ ہائی لیول پروگرامنگ پڑھیں گے اور صحیح معنوں میں پروفیشنل پروگرامنگ کلاسز سے شروع ہوتی ہے۔ اور آپ کا اوبجیکٹ اورینٹڈ پروگرامنگ کا کانسٹیٹ بھی یہاں سے شروع ہوتا ہے۔ کلاس ایک ڈرائیو ہیڈ ٹائپ ہے یعنی یہ کسی اور جز یا عنصر سے حاصل کی ہے اور اس کے عناصر کی پھر اور ٹاپکس ہوتی ہیں۔ ایک کلاس ایک ارے کی مانند ہوتی ہے لیکن ارے کے عناصر کی ٹائپ ایک ہوتی ہے جبکہ کلاسز کے عناصر کی ٹائپ مختلف بھی ہوسکتی ہے۔ اس کے علاوہ کلاسز کے عناصر ویری ایبلز کے علاوہ فنکشنز یا آپریٹر بھی ہوسکتے ہیں۔ ایسے پروگرامز جو کلاسز استعمال کرتے ہیں اسے اوبجیکٹ اورینٹڈ پروگرامنگ کہتے ہیں۔ اس میں پروگرام ایک اوبجیکٹ ماڈل بناتا ہے۔ ایک اوبجیکٹ خودمختار اینٹٹی ہے جو اپنا ڈیٹا سٹور کرتا ہے اور اس کے اپنے فنکشنز بھی ہوتے ہیں۔ اوبجیکٹ کا ایک اہم کام یہ جاننا بھی ہے کہ کس ایکشن کو کب اور کیسے پرفارم کرنا ہے۔

اوبجیکٹ میں کچھ کلازز ہوتے ہیں جنہیں ڈیٹا ممبرز کہتے ہیں اور اس میں کچھ ممبر فنکشنز بھی ہوتے ہیں۔ ممبر فنکشنز کو میتھڈز بھی کہا جاتا ہے اور یہ اس وقت کال ہوتا ہے جب اوبجیکٹ کو انسٹرکشن بھیجی جاتی ہے۔ اپنے پروگرام میں کلاسز شامل کرنا اوبجیکٹ اورینٹڈ پروگرامنگ کی طرف پہلا قدم ہوتا ہے۔

کلاس ڈیکلیریشن:

آپ کسی بھی طرح کی کلاس یوں ڈیکلیر کرتے ہیں۔

```
class first{
    public:
    void sum(void);
    void print(int, int);
    private:
    int num, den;
    };
```

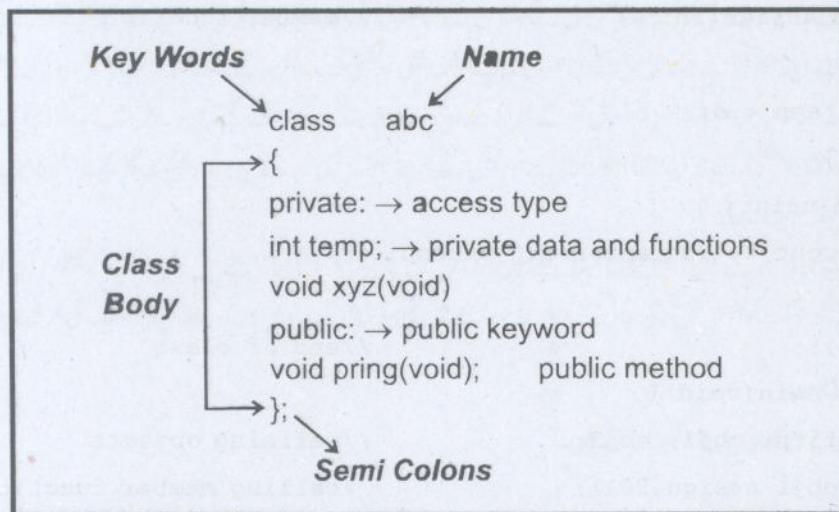
کسی بھی کلاس کا آغاز ++C کی ورڈ کلاس سے ہوگا اور اس کے بعد پھر کلاس کا نام لکھا جاتا ہے اور اس کے بعد درمیانی بریکٹ لگائی جاتی ہے اور

کلاس کا اختتام پھر بریکٹ سے ہوگا جس کے بعد سیمی کالن لکھنا ضروری ہوتا ہے۔ ہماری کلاس میں فنکشن ( )sum اور ( )print ممبر فنکشنز ہیں کیونکہ یہ کلاس کے ممبرز ہیں۔ اسی طرح وری ایبلز num اور den کلاس کے ڈیٹا ممبرز ہیں۔ آپ ممبر فنکشنز کو میتھڈز بھی کہہ سکتے ہیں۔

اس کلاس میں ہم نے میتھڈز سے پہلے public کی ورڈ لکھا ہے جبکہ ڈیٹا ممبرز سے پہلے private لکھا ہے۔ یہ ایکسیس کی ورڈز ہوتے ہیں یا انہیں ایکسیس ٹائپ بھی کہتے ہیں۔ ان دونوں میں فرق یہ ہے کہ public ممبرز اس کلاس سے باہر بھی ایکسیس (استعمال) کئے جا سکتے ہیں جبکہ private ممبرز کو آپ اس کلاس میں استعمال کر سکتے ہیں اور جب آپ کسی بھی کلاس کے ممبرز کو باہر ایکسیس کرنے کی سہولت فراہم نہیں کرتے تو اس ٹیکنالوجی کو انفارمیشن ہائیڈنگ کہا جاتا ہے ان کے بارے میں تفصیل آپ آگے پڑھیں گے۔ فی الوقت صرف ان دونوں کا فرق ذہن میں رکھیں اور کلاسز کو ایک مثال سے سمجھنے کی کوشش کریں۔

مثال نمبر 5.7 //Simple Class

```
class first{                        //class declaration
    private:
    int temp;                       //class data
    public:
void assign(int a)                  //member function
    {
    temp = a;
    }
void print( )  {
    cout <<"You entered:" <<temp;
    }
    };                              //end of class
void main(void){
    first obj1, obj2;               //defining objects
    obj1.assign(201);               //calling member functions
    obj2.assign(107);
    obj1.assign( );
    obj2.assign( );
    getch( );
    }
```

اس مثال میں ایک کلاس میں دو میتھڈز تحریر کئے ہیں جبکہ ایک ڈیٹا ممبر ہے جس کا ایکسیس موڈیفائر private ہے اور ( )main میں اس کلاس کے دو اوبجیکٹس بنائے گئے ہیں۔ اوبجیکٹ کا معمولی سا تعارف آپ نے سٹرکچر میں پڑھا اس کی تفصیل آگے پڑھیں گے۔ دو اوبجیکٹس بنانے کا یہ فائدہ ہے کہ آپ ایک فنکشن کو دو دفعہ مختلف ویلیوز کے ساتھ با آسانی کال کر سکتے ہیں۔

اوبجیکٹس:

اوبجیکٹ کا سب سے اہم جواب یہ ہے کہ اوبجیکٹ کا کلاس سے وہی تعلق ہوتا ہے جو ایک وری ایبل کا اپنی ڈیٹا ٹائپ سے ہوتا ہے۔ آپ ایک اوبجیکٹ کلاس کو ایک انسٹینس بھی کہہ سکتے ہیں۔

پرائیویٹ اور پبلک کی ورڈز:

آپ اکثر ہر ایک کے منہ سے ایک ہی لفظ سنتے ہیں کہ مجھے اوبجیکٹ اورینٹڈ پروگرامنگ کا کانسپٹ ہے۔ اس کو OOP بھی کہتے ہیں۔ OOP کا ایک اہم فیچر ڈیٹا ہائیڈنگ بھی ہے۔ اس کا یہ مطلب ہے کہ ڈیٹا کو ایک کلاس میں محدود کر دیا جائے اور کوئی دوسرا پروگرام اسے اس کلاس سے باہر ایکسیس نہ کر سکے۔ اس کے لئے ڈیٹا سے پہلے کی ورڈ private لکھا جاتا ہے۔ پرائیویٹ ڈیٹا یا میتھڈز صرف اسی کلاس میں ایکسیس کئے جا سکتے ہیں جبکہ public ڈیٹا یا میتھڈز آپ ایک کلاس سے باہر کہیں بھی استعمال کر سکتے ہیں۔

ہم نے اوپر ممبر فنکشنز کا ذکر کیا تھا۔ ایسے فنکشنز جو کلاس کی باڈی میں شامل ہوں یا جو کلاس کا ایک حصہ ہوں وہ ممبر فنکشنز کہلاتے ہیں اور اکثر ایک کلاس کے ممبر فنکشنز یا میتھڈز پبلک جبکہ ڈیٹا ممبرز پرائیویٹ ڈیکلیر کئے جاتے ہیں۔ ڈیٹا اسی لئے پرائیوٹ ڈیکلیر کیا جاتا ہے تاکہ یہ محفوظ رہ سکے اور پروگرام کا دوسرا حصہ اسے ڈسٹرب نہ کرے۔ آپ ایک کلاس کے کام کرنے کا طریقہ اس حرآرکی سے سمجھ سکتے ہیں۔

اب آپ نے کلاسز کے بارے میں کافی بنیادی معلومات حاصل کر لی ہیں۔ اس کے لئے اب ایک پروگرام لکھتے ہیں جو مزید آپ کی رہنمائی کرے گا۔

مثال نمبر 5.8

```
class employee
  {
  private:
  char name[20], address[25];
  int salary;
```

```
public:
int empno;
void input(void);
void output(void);
};
void employee::input(void)                 //function definition
{
cout <<"Employee Number";
cin >>empno;
cout <<"Enter Name";
gets(Name);
cout <<"Enter Address";
gets(Address);
cout <<"Enter Salary";
cin >>Salary;
}                                          //end of function
void employee::output(void)                //function definition
{
cout <<"Name is:" <<name;
cout <<"Address is:" <<address;
cout <<"Salary is:" <<salary;
}                                          //end of function
void main(void)
{
clrscr( );
employee emp;                              //object declaration
emp.input( );                              //funtion calling
emp.output( );
getch( );
}
```

یہاں پر ہم نے emp ایک کلاس کا اوبجیکٹ ڈیکلیر کیا ہے۔اس اوبجیکٹ کے اپنے ڈیٹا ممبرز اور میتھڈز ہیں اور یہ اوبجیکٹ اس لئے ڈیکلیر کیا ہے تاکہ یہ اپنے میتھڈز input( ) اور output( ) کو کال کرے۔آپ نے دیکھا کہ میتھڈ کال کرنے کے لئے اوبجیکٹ کا نام پھر ڈاٹ آپریٹر اور آخر پر میتھڈ

کا نام لکھا جاتا ہے۔ ایک اوبجیکٹ بالکل ایسے ہی ڈیکلیر کیا جاتا ہے جس طرح کہ ایک ویری ایبل ڈیکلیر کیا جاتا ہے لیکن فرق صرف یہ ہے کہ اس کی ڈیٹا ٹائپ کلاس ہوتی ہے اور آپ اسے اس طرح یوزر ڈیفائن ڈیٹا ٹائپ بھی کہہ سکتے ہیں۔

آپ نے دیکھا کہ جب کلاس کے میتھڈز کلاس سے باہر ڈیفائن کئے جاتے ہیں تو ان کو لکھنے کا سٹائل قدرے مختلف ہونا چاہئے۔ وہ اس لئے کہ یہ کلاس کے ممبر ہوتے ہیں اور انہیں اس کلاس کے ریفرنس سے باہر ڈیفائن کیا جاتا ہے۔ میتھڈز کی ڈیفنیشن کے لئے یہ لکھنا ضروری ہے۔ اس میں پہلے کلاس کا نام اور اس کے بعد ڈبل کالن دو دفعہ لکھا جاتا ہے۔

```
employee::input(void)
```

ان کالنز کو ریزولیشن آپریٹر (::) یا سکوپ ریزولیشن آپریٹر بھی کہتے ہیں۔

آپ نے اوپر مثال نمبر 5.8 میں ایک کلاس بنائی ہے اور اس میں آپ ایک دفعہ یوزر سے ان پٹ لے رہے ہیں یعنی آپ صرف ایک ملازم کا ڈیٹا حاصل کر رہے ہیں اور وہ پرنٹ کر دیتے ہیں۔ اگر آپ ایک سے زیادہ ملازمین کا ڈیٹا حاصل کرنا چاہتے ہیں تو اس کے لئے آپ کو یہ فنکشن مطلوبہ ٹائمز میں کال کرنا ہوگا۔ جیسا کہ ہم نے مثال نمبر 5.7 میں دو دفعہ ایک فنکشن کال کیا تھا۔ اس مثال میں ہم نے دو اوبجیکٹس بنائے تھے لیکن اگر آپ دس ملازمین کا ریکارڈ حاصل کرنا چاہتے ہیں تو کیا دس اوبجیکٹس ڈیکلیر کریں گے تو اس کا جواب یہ ہے کہ نہیں تو پھر ایسا کیسے ہوگا؟ اس کے لئے آپ اوبجیکٹ ارے ٹائپ کو ڈیکلیر کریں گے۔ آیئے اس کا حل دیکھتے ہیں۔ اس کے لئے آپ کا باقی کوڈ 5.8 مثال والا ہوگا۔ صرف آپ کو اس پروگرام کا ( )main میتھڈ تبدیل کرنا ہوگا۔ تو ہم بھی یہاں باقی کوڈ کی بجائے صرف ( )main میتھڈ دوبارہ لکھ رہے ہیں۔

مثال نمبر 5.8.1

```
//Rest of code 5.8
void main(void)
    {
    clrscr( );
    employee emp[4];
    int count=0;
    for (; cout< 4; cout ++)
    {
    cout <<endl;
    emp[count].input( );
    }
for(; count<4; count++)
    {
    cout <<endl;
    emp[count].output( );
    }
    getch( );
    }
```

اس پروگرام میں دو فنکشنز یا میتھڈز چار دفعہ کال ہو رہے ہیں کیونکہ آپ نے کلاس کا اوبجیکٹ ایک ارے بنایا ہے اور اس کو چار ویلیوز پاس کی ہیں اور لوپ کو بھی ہم نے چار دفعہ ایگزیکیوٹ کیا ہے۔ اب جب آپ اسے ایگزیکیوٹ کریں گے تو یہ چار ملازمین کا ریکارڈ پوچھے گا اور چار ملازمین کے بارے میں معلومات ڈسپلے کرے گا۔

کنسٹرکٹر:

جب ایک کلاس بنائی جاتی ہے تو اس کے ممبر کلاس کے کنسٹرکٹر فنکشن کی مدد سے اینی شلائز کیے جا سکتے ہیں۔ ایک کنسٹرکٹر کلاس کا ممبر فنکشن ہوتا ہے اور اس کا نام اور کلاس کا نام ایک ہی ہوتا ہے اور پروگرام جو کنسٹرکٹر فراہم کرتا ہے وہ جب بھی کلاس انسٹینس (instance) کا بنتا ہے خود بخود کال کر لیا جاتا ہے۔ آپ کنسٹرکٹر کو اوورلوڈ بھی کر سکتے ہیں۔ اس کا کیا طریقہ ہے آپ آگے پڑھیں گے۔

ڈیٹا ممبرز کو کلاس انسٹرکٹر میں اینی شلائز کرنا چاہئے یا ان کی ویلیوز اس وقت سیٹ کی جاتی ہے جب کلاس کا اوبجیکٹ بنایا جاتا ہے۔ آئیے کنسٹرکٹر کے لئے ایک پروگرام بناتے ہیں۔

مثال نمبر 5.9 کلاس انسٹرکٹر

```
class base
    {
    public:
    base(int a, int b)                //constructor
    {
    num=a;
    den=b;
    }
    private:
    int num, den;
    void sum( );
    };
void main(void)
    {
    clrscr( );
    base c(5, 2), f(7, 13);                     //object
    cout <<"First call c=";
    c.sum( );
    cout <<"\n Second call f=";
    f.sum( );
    getch( );
    }
```

```
void base::sum( )
{
num+=den;
cout <<num <<"+" <<den <<"=" <<num;
}
```

اس مثال میں ہم نے base نام کی ایک کلاس بنائی ہے اور اس میں ایک ( )base نام کا فنکشن بھی ہے۔ یہ اصل میں کلاس کا کنسٹرکٹر ہے اور یہ ڈیٹا ممبرز کو مخصوص ویلیوز آسائن کرتا ہے جب ( )main میں موجود ڈیکلیریشن ایگزیکیوٹ ہوگی تو کنسٹرکٹر خود بخود کال کر لیا جائے گا اور ویلیو 5 اور 2 اس میں موجود پیرامیٹرز a اور b کو پاس کر دی جائے گی۔ پھر فنکشن ( )base یہ ویلیوز num اور den کو آسائن کر دے گا۔ اصل میں ایک کلاس کا کنسٹرکٹر کلاس کے اوبجیکٹ کے لئے میموری میں سپیس ریزرو کرتا ہے آپ ایک کلاس کے ایک سے زائد کنسٹرکٹر بھی بنا سکتے ہیں یہ آپ آگے پڑھیں گے۔ آیئے اس وقت ہم ایک اور مثال لکھتے ہیں۔

مثال نمبر 5.10 کنسٹرکٹر

```
class temp
    {
    private:
    usnsigned int watch;
    public:
    temp( )
    {
    watch = 12;
    }
    };
void main(void)
    {
    clrscr( );
    temp t;
    cout <<"t.watch( )=" <<t.watch( );
    getch( );
    }
```

بعض کنسٹرکٹرز صرف ڈیٹا کو اپنی شلائز کرنے کے لئے ہی استعمال ہوتے ہیں۔ اس کے لئے ++C ایک الگ سے طریقہ فراہم کرتا ہے جسے اپنی شلائزیشن لسٹ کہتے ہیں۔ اس کے لئے آپ یہ کوڈ لکھیں گے۔

```
base(int a, int b): num(a), den(d){ }
```

اینی شلائزیشن لسٹ (:) کالن سے شروع ہوتی ہے اور فنکشن باڈی پر ختم ہوتی ہے اور فنکشن باڈی اس وقت خالی ہے۔

کنسٹرکٹر اوولوڈنگ:

ہم نے اس باب میں پہلے بھی ذکر کیا تھا کہ آپ ایک کلاس کے ایک سے زائد کنسٹرکٹر بھی بنا سکتے ہیں اس کو کنسٹرکٹر اوورلوڈنگ کہتے ہیں۔ یہ بالکل فنکشن اوورلوڈنگ کی طرح کام کرتا ہے اور C++ کمپائلر کنسٹرکٹر کے آرگومینٹ سے شناخت کرتا ہے کہ اب کون سا فنکشن کال کیا جا رہا ہے۔ آیئے اس کی ایک مثال دیکھتے ہیں۔

مثال نمبر 5.11 کنسٹرکٹر اوورلوڈ

```
class Base
    {
    Private:
    int xcord;
    float inches;
    public:
    Base( ): xcord(0), inches(0.0)
    { }                     //initilization line
    Base(int xc, float in)
    xcord=xc;
    inches=in;
    }
    Base(int xc): inches(6.4)
    {
    xcord=xc;
    }
  void getdata( )
    {
    cout <<"\n Enter x-coordinate of Rectangle in integer:";
    cin >>xcord;
    cout <<"\n Enter length in inches:";
    cin >>inches;
    }
  void display( )
    {
    double ans;
```

```
ans=(xcord+inches)/2;
cout <<"\n Answer=" <<ans;
}
};
void main( );
{
clrscr( );
Base b1(13), b2(10, 5.6), b3;
b3.getdata( );
cout <<"\n b1(13):";
b1.diplay( );
cout <<"\n b2(10, 5.6):";
b2.display( );
cout <<"\n b3.getdata( ):";
b3.display( );
getch( );
}
```

اس پروگرام میں ہم نے ایک ڈیفالٹ کنسٹرکٹر بھی لکھا ہے اس میں کوئی آرگومنٹ پاس نہیں کیا گیا یعنی base( ) کنسٹرکٹر ہماری کلاس کا ڈیفالٹ کنسٹرکٹر ہے اور جب بھی کلاس کا اوبجیکٹ بنے گا یہ خود بخود کال ہو جائے گا اس کے علاوہ اس پروگرام میں ہم نے اپنی شلائزیشن لائن بھی استعمال کی ہے اس میں ہم نے تین کنسٹرکٹر لکھے ہیں۔ ایک کو کوئی پیرامیٹر پاس نہیں کئے دوسرے میں base(int xc) صرف ایک پیرامیٹر پاس کیا ہے جبکہ تیسرے کنسٹرکٹر base(int xc, float in) کو دو پیرامیٹر پاس کئے ہیں اور کمپائلر ان پیرامیٹرز کی مدد سے یہ سمجھتا ہے کہ اب کون سا کنسٹرکٹر کال کیا جا رہا ہے۔
جب آپ اس پروگرام کو ایگزیکیوٹ کریں گے تو یہ آؤٹ پٹ ڈسپلے ہوگی۔

```
Enter x-coordinate of rectangle in integer: 12
Enter length in inches:   8.24
b1(13):
Answer = 9.7
b2(10, 5.6):
Answer = 7.8
b3.getdata( ):
Answer = 9.12
```

اوبجیکٹ بطور آرگومنٹس:

آپ فنکشن پیرامیٹرز اور آرگومنٹس سے بخوبی واقف ہیں اور آپ نے اوبجیکٹ بنانا بھی سیکھا ہے۔ ہماری اگلی مثال کچھ مختلف ہے اس میں ہم ایک فنکشن کو کلاس کا اوبجیکٹ پاس کریں گے۔ یہ کس طرح کام کرے گا آیئے اس کو سمجھتے ہیں۔

<u>مثال نمبر 5.12 اوبجیکٹ بطور فنکشن آرگومنٹس</u>

```
class Base
    {
    Private:
    int xcord;
    float length;
    public:
    Base( ): xcord(0), length(0.0)
    { }
    Base(int xc): inches(6.14)
    {
    xcord=xc;
    }
  void getdata( )
    {
    cout <<"\n Enter x-coordinate in integer:";
    cin >>xcord;
    cout <<"\n Enter length in inches(i.e 3.7):";
    cin >>length;
    }
  void display( )
    {
    cout <<"x-coordinate:" <<xcord <<"\n length:" <<length;
    double calculate(Base, Base);
    };
  double base::calculate(Base b1, Base b3);
    {
    double ans;
    length = b1.length+b3.length;     //getvalues of object b1&b3
```

```
    record = b1.xcord+b3.xcord;
    ans = (xcord+length)/2;
    return ans;                              //return answer
    }
void main(void);
    {
    clrscr( );
    Base b1(13), b3;
    double temp;
    b3.getdata( );
    cout <<"\n With default value b1(13)\n";
    b1.diplay( );
    temp = b3.calculate(b1, b3);      //value receiving
    cout <<"\n b3.display( )\n";
    b3.display( );      //after assinging new values in calculate( )
    cout <<"\n Answer of calculate( );" <<temp;
    getch( );
    }
```

آپ نے دیکھا کہ ہم نے اس پروگرام میں double calculate(Base, Base) فنکشن بنایا ہے۔ اس کو بعد میں Base ٹائپ کے دو اوبجیکٹ بھی پاس کئے گئے ہیں اور اس فنکشن کی باڈی میں ہم نے کلاس کے ڈیٹا ممبرز کو دو الگ الگ اوبجیکٹس کی مدد سے ایکسیس کیا ہے اور ان کا مجموعہ معلوم کیا ہے اور بعد میں اس کو 2 سے تقسیم کیا گیا ہے اور ان کا جواب ریٹرن کیا ہے اور یہاں سے جو ویلیو ریٹرن ہو رہی ہے اسے بطور main( ) میتھڈ میں temp ویری ایبل میں محفوظ کیا ہے۔ جب آپ اس پروگرام کو ایگزیکیوٹ کریں گے تو یہ آؤٹ پٹ ڈسپلے ہوگی۔

```
Enter x-coordinate in integer : 17
Enter length in inches (i.e 3.7) : 5.2
with default value b1(13)
x-coordinate : 13
length : 6.14
b3.display( )
x-coordinate : 30
Answer of calculate( ) : 20.67
```

فنکشن سے اوبجیکٹ ریٹرن کرنا:

آپ نے اس سے پہلے ایک پروگرام لکھا تھا جس میں اوبجیکٹ بطور آرگومنٹ پاس کیا تھا۔ اسی طرح آپ ایک فنکشن کی ریٹرن ٹائپ کسی ڈیٹا ٹائپ کی بجائے اوبجیکٹ بھی واضح کر سکتے ہیں۔ یعنی آپ کا فنکشن ایک اوبجیکٹ ریٹرن کرے گا۔ یہ کس طرح ہوگا آیئے اس کے لئے ایک پروگرام لکھتے ہیں۔

مثال نمبر 5.13 فنکشن سے اوبجیکٹ ریٹرن کرنا

```
class Super
    {
    Private:
    int num, power, fact;
    public:
    Super( ): fact(1)
    { }
void getdata( )
    {
    cout <<"Enter a positive number:";
    cin >>num;
    cout <<"\n Enter its power:";
    cin >>power;
    }
void display( )
    {
    cout <<"\n Answer:" <<fact;
    }
    Super calculate(super);
    };
Super Super::calculate(Super ans)
    {
                    //super ans;
    ans.fact=pow(num, power);
    return ans;
    }
void main(void)
```

```
{
clrscr( );
Super s1, temp;
s1.getdata( );
temp=s1.calculate(temp);
temp.display( );
getch( );
}
```

اس پروگرام میں ہم نے ایک فنکشن Super calculate(Super) بنایا ہے اس فنکشن کی ریٹرن ٹائپ کلاس ہے۔ یعنی یہاں پر ہم کلاس کا اوبجیکٹ ریٹرن کریں گے۔ یہ اوبجیکٹ ہم نے جب فنکشن کی باڈی یا ڈیفینیشن لکھی ہے تو بریکٹس میں Super Super::culculate(Super ans) ڈیکلیر کیا تھا۔ اگر آپ فنکشن کو کوئی آرگومنٹ پاس نہیں کرنا چاہتے تو Super calculate( ) کو ایسے لکھیں اور اس کی باڈی میں کلاس کا اوبجیکٹ بنالیں جیسا کہ ہم نے کومینٹس میں بنایا ہے۔

```
//Super ans;
```

بعد میں ہم نے اس اوبجیکٹ کو ریٹرن کیا ہے ;return ans اور main() میتھڈ میں ہم نے اس ریٹرن کی ہوئی ویلیو کو ایک اوبجیکٹ میں محفوظ کیا ہے۔

```
temp=s1.calculate(temp);
```

اب ہم اس حاصل کی گئی ویلیو کو ریٹرن کرنا چاہتے ہیں۔ یعنی ہم نے فنکشن calculate میں جو آپریشن پرفارم کیا ہے اسے ڈسپلے کرنا چاہتے ہیں تو اس کے لئے اس اوبجیکٹ temp کے ریفرنس سے display( ) فنکشن کال کیا ہے۔ ہمارا یہ پروگرام یوزر سے دو نمبر بطور ان پٹ لے گا اس میں دوسرا نمبر پہلے نمبر کی پاور ہوگا یعنی وہ اتنی دفعہ پہلے نمبر کو آپس میں ضرب دے گا اس کے لئے ہم نے pow( ) فنکشن استعمال کیا ہے۔

```
pow(num, power);
```

pow( ) فنکشن C++ کی <math.h> ہیڈر فائل میں ڈیفائن کیا گیا ہے۔

ڈسٹرکٹر: (Destructor):

ایک ڈسٹرکٹر ایک کلاس کا ایک سپیشل ممبر فنکشن ہوتا ہے۔ آپ نے دیکھا ہے کہ جب ایک اوبجیکٹ بنتا ہے تو ایک کنسٹرکٹر اوبجیکٹ کو منظم کرنے کے لئے خود بخود بن جاتا ہے۔ اسی طرح جب ایک اوبجیکٹ ختم ہوتا ہے تو ایک اور سپیشل ممبر فنکشن کال ہوتا ہے جو اس اوبجیکٹ کے ختم ہونے کو منظم کرتا ہے اس خصوصی فنکشن کو ڈسٹرکٹر کہتے ہیں۔ کسی بھی کلاس کے لئے ڈسٹرکٹر آپ خود بھی لکھ سکتے ہیں۔ اس میں کلاس کے نام سے پہلے ٹیلڈ (~) Tilde علامت لکھی جاتی ہے۔ یہ علامت اصل میں کمپلیمینٹ کے لئے استعمال ہوتی ہے اور ڈسٹرکٹر کلاس کے کنسٹرکٹر کا کمپلیمینٹ ہے اس لئے اس کے ساتھ یہ علامت لکھی جاتی ہے۔

ہر کلاس کا صرف ایک ڈسٹرکٹر لکھا جاتا ہے اور اگر یہ خود ڈیفائن نہ کیا جائے تو کنسٹرکٹر کی طرح یہ خود بخود کال ہو جاتا ہے۔ کنسٹرکٹر کی طرح اس کی کوئی ریٹرن ٹائپ نہیں ہوتی اور نہ ہی اس کو کوئی پیرامیٹر پاس کیا جاتا ہے۔ یہ اصل میں کلاس کے فالتو ڈیٹا ممبرز سے میموری آزاد کرواتا ہے یعنی جب کلاس کا اوبجیکٹ ختم ہو جاتا ہے تو پھر اس کے ڈیٹا ممبرز میں موجود ویلیوز ہمارے کسی کام کی نہیں ہوتیں۔ تو ان کو ختم کر دینا چاہئے تاکہ ہماری میموری مزید کسی کام کے لئے استعمال کی جا سکے یہ کام ڈسٹرکٹر کرتا ہے۔ آیئے اس کی ایک آسان سی مثال دیکھتے ہیں۔

مثال نمبر 5.14 ڈسٹرکٹر اور کنسٹرکٹر کالنگ

```
class Destructor
    {
    private:
    int temp;
    public:
    Destructer( ): temp(4)              //constructor
    {
    cout <<"\n\n constructor called:" <<temp;
    }
~Destructor( )                      //Destructor
    {
    temp = 0;
    cout <<"\n\n Destructor called:" <<temp;
    }
    };                              //end of class
void main(void)
    {
    clrscr( );
    {                               //scope of Object
    Destructor d;
    cout <<"\n now object is born:";
    }                               //scope ended
    cout <<"\n outside the range of object:";
    {                               //againg Object scope
    Destructor a;
    cout <<"again object is born:";
    }
    getch( );
    }
```

اس پروگرام کو جب آپ ایگزیکیوٹ کریں گے تو اس پروگرام کی یہ آؤٹ پٹ ہوگی۔

```
constructor called : 4
```

```
now object is born :
Destructor called : 0
outside the range of object :

Constructor called : 4
again object is born :

destructor called : 0
```

اس پروگرام میں آپ نے دیکھا کہ جونہی اوبجیکٹ کا سکوپ ختم ہوتا ہے تو ڈسٹرکٹر کال ہوتا ہے اور اس سے متعلقہ سٹیٹ مینٹ ایگزیکیوٹ ہوتی ہے۔

مشق

سوال نمبر 1: سٹرکچرز کے ممبرز کو ایکسیس کیسے کیا جاسکتا ہے؟

سوال نمبر 2: سٹرکچر استعمال کرتے ہوئے ایک پروگرام تحریر کریں جو یوزر سے دو پوائنٹس کی ویلیو لے۔ مثلاً

```
Enter coordinates for point1 = 3  4
Enter coordinates for point2 = 8  3
```

پھر آپ کا پروگرام تیسرا پوائنٹ خود معلوم کرے اور یہ پوائنٹ ان دو coordinates کو جمع کرنے سے حاصل ہوگا۔

سوال نمبر 3: ان سوالات کا مختصر جواب تحریر کریں؟

(i) C++ میں کلاس اور سٹرکچر میں کیا فرق ہے؟

(ii) کلاس کنسٹرکٹر اور ڈسٹرکٹر میں کیا فرق ہے؟

(iii) ایک کلاس کے پبلک ممبر اور پرائیویٹ ممبر میں کیا فرق ہے؟

(iv) کلاس ڈیفینیشن میں سکوپ ریزولیشن آپریٹر (::) کیوں اور کیسے استعمال کیا جاتا ہے؟

(v) کلاس کے کنسٹرکٹر اور ڈسٹرکٹر کا کیا نام ہونا چاہیے؟

سوال نمبر 4: پروگرام کے ہر حصے میں کیا ایرر ہے اور آپ اسے کیسے ختم کریں گے؟

(i)

```
class abc
    {
    public:
    int abc(const int*, int);
    private:
        //private member
    };
```

(ii) فرض کریں کہ یہ کوڈ ایک کلاس abc میں لکھا ہے۔

```
void ~abc(int, char);
```

(iii)

```
*a.temp;
```

(iv)

```
class abc
{
```

```
public:
    //member function
private:
int temp = 0;
int count = 0;
};
```

سوال نمبر5: اس پروگرام کی آؤٹ پٹ کیا ہوگی؟

```
class xyz
    {
    public:
    xyz(int a, xyz* x=0)
    {
    data = a;
    count = x;
    }
    int data;
    xyz* count;
    };
void main( )
    {
    clrscr( );
    int f;
    xyz* x;
    xyz* y;
    while(cin>>f)
    {
    x=new xyz(f, y);
    y=x;
    }
    for(;x→count; x=x→count;
    cout <<x→data<<"→";
    cout <<"\n";
    }
```

❶: جواب

جواب: سٹرکچر کے ممبرز کو ایکسیس کرنے کے لئے دو ایکسیس آپریٹرز، ڈاٹ آپریٹر (.) اور ایرو آپریٹر (→) استعمال کئے جاتے ہیں۔ ڈاٹ آپریٹر کی مدد سے آپ سٹرکچر کے اوبجیکٹ کا ریفرنس ایکسیس کرتے ہیں جبکہ ایرو آپریٹر اس وقت استعمال کرتے ہیں جب آپ اوبجیکٹ کو پوائنٹر ڈیکلیر کرتے ہیں۔ آپ اوبجیکٹ کو پوائنٹر ڈیکلیر کرتے ہیں۔ آپ اس طرح ان آپریٹرز کو استعمال کرتے ہیں۔

```
Question q2;
Question *qptr;
cout <<q2.temp;
cout <<qptr→temp;
```

❷: جواب

```
struct Question2
    {
    int xco;
    int yco;
    };
void main(void)
    {
    clrscr( );
    Question2 q1, q2, q3;
    cout <<"\n Enter coordinates for point1:";
    cin >>q1.xco >>q1.yco;
    cout <<"\n Enter coordinates for point2:";
    cin >>q2.xco >>q2.yco;
    q3.xco=q1.xco+q2.xco;
    q3.yco=q1.yco+q2.yco;
    cout <<"\n coordinates of q1+q2=" <<q3.xco.<<"," <<q3.yco;
    getch( );
    }
```

3: جواب

(i) ++C میں کلاس اور سٹرکچر تقریباً ایک ہی ہیں۔ صرف ایک نمایاں فرق ہے کہ کلاس میں ڈیفالٹ ایکسیس لیول پرائیویٹ ہوتا ہے جبکہ سٹرکچر پبلک ہوتا ہے۔

(ii) ایک کنسٹرکٹر کلاس کا ممبر فنکشن ہوتا ہے اور یہ اس وقت خود بخود ایگزیکیوٹ ہوتا ہے جب اس کلاس کا اوبجیکٹ بنایا جاتا ہے اور یہ ویری ایبلز کو انیشلائز کرتا ہے جبکہ ڈسٹرکٹر کلاس کا ایسا ممبر فنکشن ہے جو اسوقت خود بخود ایگزیکیوٹ ہوتا ہے جب کلاس کے اوبجیکٹ کا سکوپ ختم ہوتا ہے۔

(iii) پبلک ممبر کلاس سے باہر بھی ایکسیس کیے جاسکتے ہیں جبکہ پرائیویٹ ممبر کلاس سے باہر ایکسیس نہیں کیے جاسکتے۔

(iv) سکوپ ریزولیشن آپریٹر : : کسی بھی ریفرنس کے لئے استعمال کیا جاتا ہے۔ یہ کلاس کی ڈیفینیشن کلاس کے سکوپ سے باہر لکھتے ہیں۔

(v) ہر کلاس کے کنسٹرکٹر کا بالکل وہی نام ہونا چاہئے جو کلاس کا ہو جبکہ ڈسٹرکٹر کا بھی وہی نام ہونا چاہئے جو کلاس کا ہے صرف اس کے شروع میں آپریٹر (~) ٹیلڈ (Tilde) لکھا جاتا ہے۔

4: جواب

(i) کنسٹرکٹرز ویلیو ریٹرن نہیں کرسکتے۔ اس کو درست کرنے کے لئے اس کے شروع میں ریٹرن ٹائپ int ختم کردیں۔

(ii) ڈسٹرکٹر کسی بھی قسم کی ویلیو ریٹرن نہیں کرتا اور نہ ہی یہ کسی قسم کے آرگومنٹ پاس کرتا ہے۔ اس کو درست کرنے کے لئے ریٹرن ٹائپ اور پیرامیٹر int ختم کردیں۔

(iii) آپ اس طرح ڈائریکٹ ایکسیس نہیں کرسکتے کیونکہ ایکسیس آپریٹر (.) کا priority لیول سٹریک (*) سے زیادہ ہے اس کے لئے آپ کو اس کے گرد بریکٹس لگانی ہوگی۔

```
(*a).temp
```

(iv) ممبرز کلاس ڈیفینیشن میں ایکسیس نہیں کئے جاسکتے۔ اس لئے یہاں پر صرف ان کو ڈیکلیر کریں۔

5: جواب

اس پروگرام کی آؤٹ پٹ یہ ہوگی۔

```
75  71  33  12  7   ^D
7→ 12→33→71→75→
```

باب نمبر 6

آپریٹر اوورلوڈنگ

اس باب میں آپ C++ کے آپریٹرز کو کلاس اوبجیکٹ کے ساتھ استعمال کریں گے اور اس پراسیس کو آپریٹر اوورلوڈنگ کہتے ہیں اور یہ C++ کی دوسری لینگوئجز کی نسبت اضافی فیچر ہے۔ لیکن آپریٹر اوورلوڈ کرتے وقت ایک بات کا خاص خیال رکھیں کیونکہ اس کا غلط استعمال آپ کے پروگرام کو بہت مشکل بنا دے گا۔ جیسا کہ << آپریٹر C++ میں کئی آپریشنز پرفارم کرنے کے لئے استعمال ہوتا ہے اور آپ اس کو اوورلوڈ بھی کر سکتے ہیں لیکن اس بات کا خیال رکھیں کہ آپ کس کام کے لئے اس کو اوورلوڈ کر رہے ہیں۔

آپ جو بھی آپریٹر اوورلوڈ کرتے ہیں کمپائلر آپ کے آپریٹر استعمال کرنے کے مطابق اس کے لئے کوڈ لکھتا ہے۔ آپریٹرز اوورلوڈنگ کی اہمیت کو دیکھتے ہوئے ہم نے اس باب میں مندرجہ ذیل عنوانات شامل کئے ہیں۔

- فرینڈ فنکشن
- فرینڈ کلاس
- This کی ورڈ
- static ڈیٹا ممبر
- static فنکشن ممبرز
- آپریٹر اوورلوڈنگ
- آپریٹر کی ورڈ
- اوورلوڈنگ بائنری آپریٹر
- ملٹی پل اوورلوڈنگ
- مشق

فرینڈ فنکشن: (Friend Function):

آپ نے اس کتاب کے باب نمبر 5 میں پڑھا ہوگا کہ کوئی بھی نان ممبر فنکشن اوبجیکٹ کے پرائیویٹ ڈیٹا ممبرز کو ایکسیس نہیں کر سکتا۔ یعنی کہ اگر آپ ممبر فنکشن نہیں تو آپ پرائیویٹ یا پروٹیکٹڈ (Private or Protected) ڈیٹا ممبر کو ایکسیس نہیں کر سکتے۔

فرینڈ فنکشن بھی ایک نان ممبر فنکشن ہوتا ہے۔ آپ سوچ رہے ہوں گے کہ پھر اس کا کیا فائدہ ہے۔ تو فرینڈ فنکشن ایسا نان ممبر فنکشن ہے جو کلاس میں ڈیکلیر کے لئے تمام ڈیٹا ممبرز کو ایکسیس کر سکتا ہے خوہ وہ پرائیویٹ ہوں یا پروٹیکٹڈ۔ تو یہ اس کا سب سے بڑا فائدہ ہے کہ ایک کلاس کا ممبر فنکشن نہ ہوتے ہوئے بھی یہ کلاس کے تمام ڈیٹا ممبرز کو ایکسیس کر سکتا ہے۔ یہ زیادہ تر آپریٹر اوور لوڈنگ میں استعمال ہوتا ہے۔ اس فنکشن کو کیسے بناتے ہیں آیئے اس کے لئے ایک پروگرام لکھتے ہیں۔

مثال نمبر 6.1 فرینڈ فنکشن

```
class abc;                              //need for friend function
class xyz {
    friend int function(xyz,. abc);     //friend function
    private:
    int temp;
    public:
    xyz(int a)
    {
    temp = a;
    }
    };
class abc
    {
    friend int function(xyz, abc);
    private:
    int temp;
    public:
    abc(int a)
    {
        temp = a;
    }
    };
int function(xyz x, abc a)
```

```
    {
    return(x.temp* a.temp);
    }
void main(void)
    {
    clrscr( );
    xyz x(4);
    abc a(7);
    cout <<"\n Answer x(4), a(7)=" <<function (x,a);
    getch( );
    }
```

اس پروگرام میں دو کلاسز xyz اور abc بنائی گئی ہیں۔ ہم نے سب سے اوپر ;class abc ڈیکلیر کی ہے وہ اس لئے کہ اس کو فرینڈ فنکشن میں استعمال کیا گیا ہے اور آپ دیکھیں گے کہ فرینڈ فنکشن کی ورڈ friend سے ڈیکلیر کیا گیا ہے اور یہ دونوں کلاسز کا ممبر فنکشن نہیں ہے۔ اس پروگرام میں دونوں کلاسز کے کنسٹرکٹر کو ایک ایک آرگومنٹ پاس کیا گیا ہے اور فرینڈ فنکشن میں دونوں کلاسز کے پرائیویٹ ڈیٹا ممبر (;int temp) کو استعمال کیا گیا ہے۔ اس پروگرام کی آؤٹ پٹ کچھ یوں ہوگی۔

```
Answer x(4), a(7) = 28
```

ہم نے فرینڈ فنکشن کو دو پیرامیٹر پاس کئے ہیں جو کہ کلاس کے اوبجیکٹس ہیں۔ آپ اس فنکشن سے پہلے لکھا گیا کی ورڈ friend ختم کریں اور پھر اس پروگرام کو چلائیں تو یہ ایرر دے گا۔

```
xyz::temp is not accessible
abc::temp is not accessible
```

فرینڈ کلاس:

اوپر آپ نے ایک فرینڈ فنکشن بنانا سیکھا۔ آپ اس کے علاوہ آپ ایک کلاس کے تمام ممبر فنکشن ایک ہی دفعہ فرینڈز بھی بنا سکتے ہیں لیکن یہ اس وقت ممکن ہوگا جب آپ ایک کلاس کو فرینڈ ڈیکلیر کریں گے۔ یہ کس طرح ممکن ہے آیئے دیکھتے ہیں۔

<u>مثال نمبر 6.2 فرینڈ کلاس</u>

```
class temp
    {
    friend class second;
    private;
    int a;
    int function( )
    {
```

```
return a*a;
}
public:
temp (int x)
{
a=x;
}
};
class second
{
public:
void sec1(temp t)
{
cout <<"\n private data member value=" <<t.a;
cout <<"\n private member function value=" <<t.func( );
}
};
void main(void)
clrscr( );
int x;
cout <<"\n Enter parameter value of object:";
cin >>x;
temp tp(x);
second sd;
sd.sec1(tp);
getch( );
}
```

اس پروگرام میں ہم نے دو کلاسز temp اور second بنائی ہیں اور second کلاس کو فرینڈ ڈیکلیر کیا ہے۔ اب یہ دوسری کلاس second پہلی کلاس کے تمام ڈیٹا ممبرز کو ایکسیس کر سکتی ہے۔ ہم نے temp کلاس کا کنسٹرکٹر لکھا ہے جس کو ایک پیرامیٹر پاس کیا گیا ہے اور اس کی ویلیو ہم یوزر سے لے رہے ہیں۔ کلاس second میں ( )sec1 نام کا ایک فنکشن لکھا ہے جس کو کلاس temp کا اوبجیکٹ پاس کیا ہے اور اس کی مدد سے temp کلاس کے تمام پرائیویٹ ڈیٹا ممبرز ایکسیس کئے ہیں۔ اس پروگرام کی آؤٹ پٹ یہ ہوگی۔

```
Enter parameter value of object : 8
private data member value = 8
```

```
private member function value = 64
```

This کی ورڈ:

کلاس کا ہراوبجیکٹ اپنے ایڈریس کو بھی ایکسیس کر سکتا ہے۔اس کے لئے this پوائنٹر استعمال کیا جاتا ہے اور یہ this پوائنٹر اوبجیکٹ کا حصہ نہیں ہوتا۔آپ یہ پوائنٹر اوبجیکٹ ڈیٹا ممبر اور ممبر فنکشن دونوں کے لئے استعمال کر سکتے ہیں۔اس پوائنٹر this کی ٹائپ کیا ہوگی؟ یہ اوبجیکٹ کی ٹائپ پر منحصر ہے اس پوائنٹر کو آپ کیسے استعمال کر سکتے ہیں۔آیئے دیکھتے ہیں۔

مثال نمبر 6.3 This پوائنٹر

```
class exthis
    {
    public:
    exthis(int);
void output( );
    private:
    int temp;
    };
exthis::exthis(int t)
    {
    temp = t*t;
    }
void exthis::output( )
    {
    cout <<"\n temp=" <<temp;
    cout <<"\ this->temp=" <<this->temp;
    cout <<"\n this, temp=" <<(*this).temp;
    }
void main( )
    {
    clrscr( );
    exthis th(11);
    th.output( );
    }       //end main
```

static ڈیٹا ممبر:

کلاس کے ڈیٹا ممبرز کی ہر کلاس اوبجیکٹ کے لئے الگ الگ کاپی ہوتی ہے یعنی کہ تمام اوبجیکٹس کے پاس ڈیٹا کی تفصیل الگ الگ ہوتی ہے۔ لیکن بعض اوقات تمام ویری ایبلز کی صرف ایک کاپی تمام اوبجیکٹس کو فراہم کرنے کی ضرورت بھی پیش آ جاتی ہے۔ ایسی صورتِ حال کا سامنا کرنے کے لئے کلاس ویری ایبلز کو static. ڈیکلیر کیا جاتا ہے۔ ایسے ویری ایبلز صرف ایک دفعہ اپنی شلائز کئے جاتے ہیں۔ یہ ویری ایبل کس طرح ڈیکلیر کئے جاتے ہیں۔ آیئے اس کی مثال دیکھتے ہیں۔

مثال نمبر 6.4 static کلاس ڈیٹا ممبرز

```
class values
{
public:
values( )
{
++temp;
}
static int temp;                //declaring static variable
};
int values::temp = 2;           //initilization
void main(void)
{
values val1, val2
cout <<"\n now value=" <<val1.temp;
{
values val1, val2, val3
cout <<"\n now value of temp=" <<val1.temp;
}
cout <<"\n Again value=" <<val1.temp;
values val3;
cout <<"\n here value of temp=" <<val3.temp;
getch( );
}
```

static فنکشن ممبرز:

آپ نے مثال نمبر 6.1 میں کلاس ویری ایبلز کو static ڈیکلیر کرنے کا طریقہ سیکھا ہے اس طرح آپ کلاس فنکشنز کو بھی static ڈیکلیر کر سکتے

ہیں۔ آیئے اس کی ایک مثال دیکھتے ہیں۔

مثال نمبر 6.5 static کلاس فنکشنز

```
//static class functions
class example
    {
    public:
    example( )              //Constructor
    {
    ++temp;
    }
    ~example( )             //Destructor
    {
    --temp;
    }
    static intfact( )
    {
    return temp;
    }
    private:
    static int temp;
    };
int example::temp=1;
void main(void)
    {
    clrscr( );
    cout <<"value of temp=" <<example::fact( );
    example obj1, obj2;
    cout <<"\n temp=" <<example::fact( );
    }
    cout <<"\n temp outside scope=" <<example::fact( );
    example obj3, obj4;
    cout <<"\n temp=" <<example::fact( );
```

آپریٹراوورلوڈنگ:

اس سے پہلے آپ نے فنکشن اوورلوڈنگ کے بارے میں پڑھا تھا۔ لیکن یہاں پر ہم فنکشن کی بجائے آپریٹرز کا ذکر کر رہے ہیں۔ ++C آپ کو تقریباً 45 آپریٹرز استعمال کرنے کی اجازت دیتا ہے۔ یہ آپریٹرز ++C کی بنیادی ٹائپس کے لئے پہلے سے ڈیفائن کئے گئے ہوتے ہیں لیکن جب آپ ایک کلاس کی بات کرتے ہیں یعنی ایک کلاس ڈیفائن کرتے ہیں تو آپ ایک نئی ٹائپ بنا رہے ہوتے ہیں۔ ++C کے بعض آپریٹرز اس نئی ٹائپ کو استعمال کرتے ہوئے اوورلوڈ کئے جاتے ہیں۔

درحقیقت آپریٹراوورلوڈنگ آپ کو ++C لینگوئج کو دوبارہ ڈیفائن کرنے کی ایک سہولت فراہم کرتا ہے۔ آپ سمجھتے ہیں کہ ++C آپریٹرز کی مدد سے آپ صرف محدود کام کر سکتے ہیں تو ایسا نہیں ہے بلکہ آپ ان کے کام کو اپنی ضرورت یا مرضی کے مطابق تبدیل کر سکتے ہیں۔

آپریٹر کی ورڈ: (Operator Keyword):

آپ نے آپریٹراوورلوڈ کرنے کا فیصلہ کر لیا ہے لیکن ایک سوال آپ کے ذہن میں ہوگا کہ ++C کمپائلر کو کیسے پتہ چلے گا کہ ہم کون سا آپریٹراوورلوڈ کرنا چاہتے ہیں یعنی ہم ایک ++C آپریٹر کو کیسے یوزر ڈیفائن آپریٹر بنا سکتے ہیں۔ اس کے لئے ++C ہمیں ایک کی ورڈ فراہم کرتی ہے جس کو استعمال کرتے ہوئے ہم مطلوبہ آپریٹر کی نشاندہی کر سکتے ہیں اور یہ کی ورڈ آپریٹر (Operator) ہے۔

آیئے آپریٹراوولوڈنگ کی ایک مثال دیکھتے ہیں۔ یہ بہت سادہ مثال ہے اس میں ہم یونری آپریٹراوورلوڈ کریں گے۔

مثال نمبر 6.6 آپریٹر کی ورڈ

```
class Example
    {
    private:
    int temp;
    public:
    Example( )
    {
    temp = 3;
    }
    int output( )
    {return temp;}
Example operator++( )
    ++temp;
    Example over;
    over.temp=temp;
    return over;
    }
    };
```

```
void main(void)
    {
    clrscr( );
    Example obj1, obj2;
    cout <<"\n obj1=" obj1.output( );
    cout <<"\n obj2=" obj2.output( );
    ++obj1;
    obj2=++obj1;
    cout <<"\n obj1=" obj1.output( );
    cout <<"\n obj2=" obj2.output( );
    getch( );
    }
```

اوورلوڈنگ بائنری آپریٹرز:

آپ نے اوپر مثال نمبر 6.6 میں یونری آپریٹرز اوورلوڈ کرنے کا طریقہ دیکھا اسی طرح آپ بائنری آپریٹرز بھی اوورلوڈ کر سکتے ہیں۔ بائنری آپریٹرز کون کون سے ہیں اس کے بارے میں آپ تفصیل سے باب نمبر 1 اور 2 میں پڑھ چکے ہیں۔ آیئے سب سے پہلے آرتھمیٹک آسائنمنٹ آپریٹرز اوورلوڈ کرنے کا طریقہ دیکھتے ہیں۔

مثال نمبر 6.7 اوورلوڈنگ آرتھمیٹک آسائنمنٹ آپریٹر

```
class exoverload
    {
    int x,y;
    public:
    exoverload( )
    {
    x=0;
    y=0;
    }
exoverload(int a, int b)
    {
    x=a;
    y=b;
    }
```

```
viod input( )
    {
    cout <<"\n Enter two integer values i.e 2,5:";
    cin >>x >>y;
    }                          //End of function
void output( )
    {
    cout <<"\n x=" <<x <<"\n y=" <<y;
    }                          //End of output
void operator*=(exoverload);
    };                         //End of class
void exovererload::operator*=(exoverlod over)
    {
    x*=over.x;
    y*=over.y;
    }
void main(void)
    {
    clrscr( );
    exoverload over1;
    over1.input( );
    cout <<"\n data output:\n";
    over1.output( );
    exoverload over2(13, 5);
    cout <<"\n data with constructor passing parameter:";
    over2.output( );
    over1*=over2;
    cout <<"\n after overloading:";
    over2.output( );
    getch( );
    }
```

اس پروگرام میں ہم نے =* آپریٹر اوورلوڈ کیا ہے۔ ایک کلاس ڈیفائن کی ہے جس کے دو کنسٹرکٹر ہیں۔ ایک کنسٹرکٹر میں کوئی پیرامیٹر پاس نہیں کیا

جبکہ دوسرا کنسٹرکٹر دو پیرامیٹرز پاس کرتا ہے۔ اس کے علاوہ ہم نے input( ) فنکشن میں یوزر سے دو نمبر بطور ان پٹ لئے ہیں اور (exoverload)=*operator میں ہم نے اس آپریٹر کواوورلوڈ کیا ہےاورایک اوبجیکٹ میں اس کا رزلٹ سٹور کروایا ہے۔

ملٹی پل اوورلوڈنگ:

آپ نے اس سے پہلے یونری اور بائنری آپریٹرز اوورلوڈ کرنے کا طریقہ پڑھا ہے۔ اب ہم آپ کو ایک ہی پروگرام میں دو یا دو سے زائد آپریٹرز اوورلوڈ کرنے کے بارے میں بتائیں گے۔ یہاں پر آپ آرتھمیٹک اور کمپریژن آپریٹرز اوورلوڈ کرنے کا طریقہ دیکھیں گے۔

مثال نمبر 6.8 کمپریژن آپریٹر اوورلوڈنگ

```
class overloading
    {
    private:
    int x,y;
    public:
    overloading( )
    {
    x=4;
    y=7;
    }
overloading(int a, int b)
    {
    x=a;
    y=b;
void input( )
    {
    cout <<"Enter two integer values i.e 2,5:";
    cin >>x >>y;
    }
void output( )const
    {
    cout <<"\n x=" <<x <<"\n y=" <<y;
    }
intput operator>(overloading);
    };
int overloading::operator>(overloading over)
```

```
    {
    int res1=(x*y)/3;
    int res2=(over.x*over.y)/3;
    if(res1>res2)
    return 1;
    else
    return 0;
    }
void main()
    {
    clrscr
    overloading over1;
    over1.input( );
    overloading over2(13, 5);
    cout <<"\n Data output:\n";
    over1.output( );
    //cout <<"\n Data with argument constructor";
    over2.output( );
    if(over1>over2)
    cout <<"\n over 1 is greater than over2:";
    else
    cout <<"\n over 1 is less than over2:";
    //over1.output( );
    getch( );
    }
```

ہم نے اس پروگرام میں ایک فنکشن int operator>(overloading) بنایا ہے۔ یہ ایک int ویلیو ریٹرن کرتا ہے جو کہ if(res1>res2) سٹیٹ مینٹ سے حاصل ہو رہی ہے۔

سوال نمبر 1: ان سوالات کے مختصر جواب تحریر کریں۔

(i) this* کس چیز کو ریفر کرتا ہے؟

(ii) ان دونوں ڈیکلیریشن میں کیا فرق ہے؟

```
xyz x(a);
xyz x=a;
```

(iii) آپ اس آپریٹر '**' کیوں اوورلوڈ نہیں کر سکتے ؟

(iv) آرتھمیٹک آپریٹرز (+, -, *, /) فرینڈ فنکشن کے طور پر کیوں اوورلوڈ کئے جاتے ہیں؟

(v) آپریٹر کی ورڈ کیسے استعمال کیا جاتا ہے۔

سوال نمبر 2: آپریٹر کو اوورلوڈ کرتے ہوئے اس مساوات کو حل کریں؟ $\frac{15}{9}+1$

سوال نمبر 3: پروگرام کے اس کوڈ میں کیا ایرر ہے؟ فرض کریں کہ یہ **xyz** کلاس میں لکھا ہے۔

```
xyz& xyz::operator=(const xyz&r)
{
x=r.x;
y=r.y;
return &this;
}
```

جوابات

1: جواب

(i) کی ورڈ this اوبجیکٹ کے لئے ایک پوائنٹر ہوتا ہے اور جو اپنے ممبر فنکشن کو خود کال کرتا ہے یا یہ اوبجیکٹ کو ریفر کرتا ہے اور یہ اس کے ممبر فنکشن کال کرتا ہے جس میں یہ ایکپسریشن لکھی ہو۔

(ii) اس میں پہلی ڈیکلیریشن ;xyz x(a) ڈیفالٹ کنسٹرکٹر کو کال کرتی ہے جبکہ ڈیکلیریشن ;xyz x=a کاپی کنسٹرکٹر کو کال کرتی ہے۔

(iii) یہ علامت ** آپریٹر کی طرح اوورلوڈ نہیں ہوسکتی کیونکہ یہ ++C آپریٹر نہیں ہے۔

(iv) آرتھمیٹک آپریٹرز فرینڈ فنکشن کے طور پر اس لئے اوورلوڈ کئے جاتے ہیں تاکہ ان کے بائیں طرف والے آپرینڈ کانسٹنٹ ڈیکلیر کئے جاسکیں۔

(v) آپریٹر کی ورڈ فنکشن اس لئے استعمال ہوتا ہے کہ یہ اس فنکشن کی نشاندہی کرتا ہے جو کسی آپریٹر کو اوورلوڈ کر رہا ہوتا ہے۔ مثلاً "*operator" اب کمپائلر کو معلوم ہوگیا ہے کہ یہ فنکشن * آپریٹر کو اوورلوڈ کرنے کے لئے استعمال کیا گیا ہے۔

2: جواب

solving $\frac{15}{9}-1$

```
class question
    {
    friend ostream& operator <<(ostream&, const question2);
    public:
    question(int a=0, int c=1)
    {
    x=a;
    d=c;
    }
    question operator ++( );
    private:
    int n,d;
    };
question2 question2::operator ++( );
    {
    n+=d;
    retrun *this;
```

```
    }
    ostream& operator <<(ostream&.str, const question2&q)
    {
    return str <<q.n <<'/' <<q.d;
    }
void main(void)
    {
    clrscr( );
    question2 q(15, 9), y=++q;
    cout <<"Answer of 15/q+1=" <<q;
    getch( );
    }
```

❸: جواب

اس میں آپ نے کی ورڈز this استعمال کیا ہے جو کہ اوبجیکٹ کے لئے ریفرنس نہیں ہوتا اس کو یوں لکھنا ہوگا۔

```
    return *this;
```

باب نمبر 7

انہریٹینس اینڈ پولی مار فیزیم

اس باب میں ہم اوبجیکٹ اوورینٹڈ پروگرامنگ کے دو اہم فیچرز انہریٹینس اور پولی مارفزیم کے بارے میں آپ کو بتائیں گے۔ انہریٹینس میں آپ ایک سافٹ ویئر کو بار بار استعمال کرتے ہیں۔ اس میں نئی کلاسز پہلے سے بنی ہوئی کلاسز سے ڈرائیوڈ کی جاتی ہیں اور ان نئی کلاسز میں پہلی کلاس کی تمام وہ خصوصیات شامل ہوتی ہیں جن کی وہ اجازت دیتی ہے۔ ایک ہی سافٹ ویئر کو بار بار استعمال کرنے سے آپ کا کوڈ آسان رہتا ہے اور اس میں ٹائم کی بچت ہوتی ہے۔ اس باب میں آپ مندرجہ ذیل عنوانات پڑھیں گے۔

- کمپوزیشن
- انہریٹینس
- انہریٹینس کی اقسام
- اووررائیڈنگ ممبر فنکشن
- کنسٹرکٹر اور ڈسٹرکٹر
- ملٹی پل کلاسز
- ورچوئل فنکشن
- Abstract کلاسز
- ورچوئل ڈسٹرکٹر
- انہریٹینس کے لیولز
- ملٹی پل انہریٹینس
- فائل پروسیسنگ
- مشق

کمپوزیشن: (Composition):

یہ اوبجیکٹ اوورینٹڈ (object oriented) پروگرامنگ کا ایک اہم فیچر ہے۔ہمیں اکثر اپنی پہلے سے بنی ہوئی کلاسز کی مدد سے نئی کلاسز بنانی ہوتی ہیں۔اس کے لئے C++ دوطریقے فراہم کرتا ہے۔کمپوزیشن اور انہریٹینس۔اس باب میں آپ یہ دونوں طریقے پڑھیں گے۔لیکن پہلے ہم آپ کو کمپوزیشن کے بارے میں بتاتے ہیں۔

کمپوزیشن میں آپ ایک کلاس کی ڈیفینیشن میں ایک یا ایک سے زائد کلاسز استعمال کرتے ہیں۔جب آپ کی نئی کلاس کا کوئی ڈیٹا ممبر پہلی کلاس کا اوبجیکٹ ہوتا ہے تو ہم کہتے ہیں کہ نئی کلاس کسی دوسرے اوبجیکٹ کی کمپوزٹ ہے۔آپ یہ کام کس طرح کر سکتے ہیں۔آیئے اس کی ایک مثال دیکھتے ہیں۔

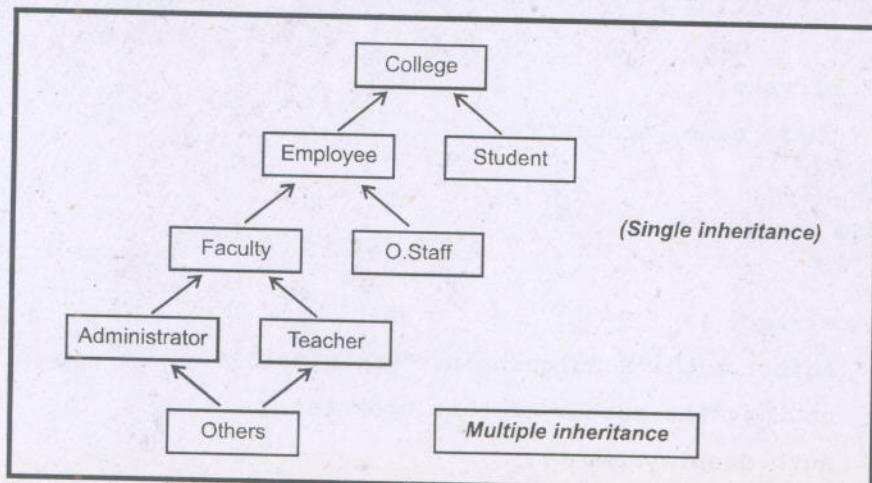
مثال نمبر 7.1

```
#include<iostream.h>
#include<string.h>
#include<iomanip.h>
class Author
    {
    public:
    Author(char* a=" ", char* cou=" ")
    :name(a), country(cou){}
void display name( )
    {
    cout <<name;
    }
void display country( )
    {
    cout <<country;
    }
    pirvate:
    char* name, *country;
    };
void main(void)
    {
    clrscr( );
    Author auth("M.Zulqurnain, "Pakistani");
    cout <<"The author of this book is";
    auth.displayname( );
```

```
cout <<setw(2) <<"and he is a"";
auth.displaycountry( );
getch( );
}
```

جب آپ اس پروگرام کو ایگزیکیوٹ کریں گے تو یہ آؤٹ پٹ مانیٹر سکرین پر ظاہر ہوگی۔

```
The author of this book is M.Zulquranin and he is a Pakistani.
```

انہریٹینس: (Inheritance):

انہریٹینس اوبجیکٹ اووریئنٹڈ پروگرامنگ کا سب سے پاورفل فیچر ہے۔ پہلے سے بنی ہوئی کلاسز سے نئی کلاسز بنانے کے پراسس کو انہریٹینس کہتے ہیں اور نئی کلاسز ڈرائیوڈ (Derived) کلاسز کہلاتی ہیں۔ ڈرائیوڈ کلاس میں اپنی بیس کلاس کی تمام خصوصیات ہوتی ہیں۔ اس کے علاوہ اس میں اپنی اضافی معلومات بھی ہوسکتی ہیں۔ یعنی انہریٹینس کی مدد سے آپ پہلے سے بنے ہوئے سافٹ ویئرز کو استعمال کرتے ہوئے نئے سافٹ ویئر بنا سکتے ہیں۔ انہریٹینس کی مدد سے آپ ایک کوڈ بار بار استعمال کر سکتے ہیں۔ مثلاً آپ نے ایک کلاس بنائی ہے اس کو ٹیسٹ کیا کہ کیا یہ درست کام کر رہی ہے یا نہیں؟ اب آپ کو جہاں بھی ایسا کام کرنا ہے تو اس کلاس کو انہریٹ کریں اور نئی کلاس میں اس کے ڈیٹا ممبرز استعمال کریں۔ اس سے آپ کا ٹائم بچ جاتا ہے۔ اس کے علاوہ بجٹ بھی متاثر نہیں ہوتا اور اہم بات یہ کہ پروگرام سادہ رہتا ہے۔ آپ دو اقسام کی کلاسز بنا سکتے ہیں۔

(i) سنگل کلاس انہریٹینس (ii) ملٹی پل کلاس انہریٹینس

سنگل انہریٹینس میں ایک کلاس صرف ایک بیس کلاس سے ڈرائیوڈ کی جاتی ہے جبکہ ملٹی پل انہریٹینس میں ایک کلاس کئی کلاسز سے ڈرائیوڈ کی جاتی ہے۔ ہم نے اوپر بھی بتایا تھا کہ ڈرائیوڈ کلاس میں اپنی بیس (base) کلاس کے علاوہ اپنے ڈیٹا ممبرز اور ممبر فنکشنز بھی ہو سکتے ہیں۔ یوں ایک ڈرائیوڈ کلاس اپنی بیس کلاس سے بڑی بھی ہوسکتی ہے۔ ایک ڈرائیوڈ کلاس میں اوبجیکٹس بیس کلاس کی نسبت بہت چھوٹے ہوتے ہیں۔

سپر کلاس اور ڈرائیوڈ کلاس:

جب آپ ایک کلاس سے دوسری کلاس کو ڈرائیوڈ کریں گے تو ایک کلاس کا اوبجیکٹ دوسری کلاس میں شامل ہوتا ہے اور اکثر ایک کلاس کا اوبجیکٹ دوسری کلاس کا بھی اوبجیکٹ ہوتا ہے۔ انہریٹینس کس طرح کام کرتی ہے۔ اس کا ڈایاگرام ہم نے نیچے بنایا ہے۔

اوپر حرآر کی میں سٹوڈنس کلاس کالج کا حصہ ہے یعنی آپ یہ کہہ سکتے ہیں کہ سٹوڈنٹ کلاس کالج کلاس سے انہریٹ کی گئی ہے۔اس کیس میں کلاس کالج بیس(base) کلاس ہے جبکہ سٹوڈنٹس ڈرائیوڈ کلاس ہے۔آپ نے انہریٹینس کے بارے میں ابھی پڑھا ہے۔آیئے اب اس کی ایک سادہ سی مثال دیکھتے ہیں کہ C++ میں ہم ایک کلاس کی خصوصیات دوسری کلاس میں کیسے انہریٹ کر سکتے ہیں۔

مثال نمبر 7.2 کلاس انہریٹینس پروگرام

```
#include<conio.h>
#include<iostream.h>
class base
    {
    Private
    int temp;
    public:
    base(int);                  //constructor
    void print(void);
    };
class derived:public base
    {                           //class inherited from base
    private:
    int count;
    public:
    derived(int, int);          //derived class constructor
    void show(void);
    };
void base::base(int a)          //base class constructor
    {
    temp=a;
    }
void derived::derived(int c, int d)
                                //derived class constructor
    :base(d)                    //call to base constructor
    {
    count=c;
```

```
    }
void base::print( )
    cout <<"\n the initilized value=" <<temp;
    }
void derived::show( )
    {
    cout <<"\n Derived class:";
    cout <<"\n The initilized value=" <<count;
    print( );
    }
void main(void)
    {
    clrscr( );
    derived obj(12, 17);            //object of derived class
    obj.show( );
    getch( );
    }
```

جب آپ اس پروگرام کو ایگزیکیوٹ کریں گے تو آپ کو یہ آؤٹ پٹ ملے گی۔

```
Derived class:
The initilized value = 12
The initilized value = 17
```

اس پروگرام میں ایک کلاس base ہے جس کا ڈیٹا ممبر temp پرائیویٹ ڈیکلیر کیا گیا ہے اور ممبر فنکشنز میں کنسٹرکٹر اور پرنٹ پبلک ڈیکلیر کئے گئے ہیں۔ اس کے بعد ایک کلاس derived کے نام سے بنائی گئی ہے جو base کلاس سے انہریٹ کی گئی ہے۔ یعنی base کلاس کی تمام خصوصیات اس میں موجود ہیں۔ اس کلاس میں count وری ایبل (ڈیٹا ممبر) پرائیویٹ جبکہ اس کلاس کا کنسٹرکٹر اور show( ) فنکشن پبلک ڈیکلیر کئے ہیں۔ اس ڈرائیوڈ کلاس کے کنسٹرکٹر میں ہم نے بیس کلاس کو اپنی شلائز کرنے کے لئے اس کا کنسٹرکٹر کال کیا ہے۔

```
void derived::derived(int c, int d)
:base(d);
```

اس کے علاوہ پرنٹ فنکشن میں وہ وری ایبل پرنٹ کیا ہے جو base کلاس کے کنسٹرکٹر میں اپنی شلائز کیا گیا ہے جبکہ show( ) میں ڈرائیوڈ کلاس میں اپنی شلائز کیا گیا وری ایبل پرنٹ کروایا ہے اور بیس کلاس کا پرنٹ فنکشن بھی show( ) میں کال کیا گیا ہے۔

انہریٹینس کی اقسام:

اوپر آپ نے سادہ پروگرام لکھا جس میں ایک کلاس سے پہلے سے بنی ہوئی کلاس کی مدد سے بنائی گئی۔یعنی یہ انہریٹینس کی ایک سادہ مثال ہے۔ C++ تین قسم کی انہریٹینس پر فارم کرنے کی سہولت فراہم کرتا ہے۔

(i) Public (ii) Private (iii) Protected

آپ پبلک اور پرائیویٹ کے بارے میں باب نمبر 5 تفصیل سے پڑھ چکے ہیں۔اس باب میں ہم آپ کا تعارف ایک نئے کی ورڈ protected سے کروائیں گے۔ پبلک انہریٹینس میں ڈرائیوڈ کلاس کا ہر او بجیکٹ بیس کلاس کا بجیکٹ تصور کیا جاتا ہے لیکن یہ درست نہیں ہے اور ایک ڈرائیوڈ کلاس اپنی بیس کلاس کے پرائیویٹ ممبرز ایکسیس نہیں کر سکتی۔لیکن ایک ڈرائیوڈ کلاس اپنی سب کلاس کے protected ڈیٹا ممبرز ایکسیس کر سکتی ہے۔آیئے اس کے لئے ایک پروگرام بناتے ہیں۔

مثال نمبر 7.3

```
class age
    {
    protected:
    int a;                          //protected data members
    char v[25];
    public:
    age(int);
                                    //class inheritance
    };
 class name:public age
     {
     public:
     void show( );
     name( );
     };
  void age::age(int v)
     {
     a=v;
     }
  void name::name( )
                                    //base class constructor
     :age(19)
     {
     cout <<"\n Enter your name:";
```

```
    cin >>v;
    }
void name::show( )
    {
    cout <<"\n Your age:" <<a;
    cout <<"\n Your name:" <<v;
    }
void main( )
    {
    clrscr( );
    name n;                  //object of derived class
    n.show( );
    getch( );
    }
```

اس پروگرام میں ہم نے کلاس age کے دو ڈیٹا ممبرز کو protected ڈیکلیر کیا ہے اور ان کو پھر مزید ڈرائیوڈ کلاس name میں استعمال کیا ہے۔ اگر آپ ان ڈیٹا ممبرز کو پرائیویٹ ڈیکلیر کرتے ہیں تو یہ ایرر دے گا۔ کیونکہ ایک کلاس کے پرائیویٹ ڈیٹا ممبرز کو دوسری کلاس ایکسیس نہیں کر سکتی۔ جب آپ اس پروگرام کو ایگزیکیوٹ کریں گے تو یہ آؤٹ پٹ ڈسپلے ہوگی۔

```
Enter your name : Cristina Maria
Your age : 19
Your name : Cristina Maria
```

اوورائیڈنگ ممبر فنکشنز: (Overriding Member Functions):

آپ نے اس سے پہلے بیس کلاس کے صرف ڈیٹا ممبرز (ویری ایبلز وغیرہ) ڈرائیوڈ کلاس میں استعمال کئے ہیں۔ آپ اس کے علاوہ بیس کلاس کے ممبر فنکشنز بھی ڈرائیوڈ کلاس میں استعمال کر سکتے ہیں۔ اور ان کا نام وہی ہوگا جو بیس کلاس میں ہوگا۔ اس میں ایک بات بڑی اہم ہے کہ اگر آپ بیس کلاس کا ممبر فنکشن بالکل اسی نام اور پیرامیٹرز کے ساتھ ڈرائیوڈ کلاس میں ڈیکلیر کر رہے ہیں تو یہ فنکشن اوورائیڈ کہلائے گا لیکن اگر آپ ڈرائیوڈ کلاس میں فنکشن کے پیرامیٹر تبدیل کر دیتے ہیں تو یہ فنکشن اوورلوڈنگ ہوگی۔ اس کے بارے میں آپ باب نمبر 2 میں پڑھ چکے ہیں۔ اس لئے یہاں پر ہم صرف اوورائیڈنگ کا ذکر کریں گے۔

فرض کریں کہ ایک کلاس b آپ نے کلاس a سے ڈرائیو کی ہے اور a میں ایک فنکشن sum( ) ہے اور یہی فنکشن آپ نے b کلاس میں بھی ڈیکلیر کیا ہے تو یہ فنکشن اوورائیڈنگ ہے۔ اب آپ نے فنکشن sum( ) کال کرنا ہے تو کس طرح آپ کون سا فنکشن کال کر سکتے ہیں اس کا ایک مخصوص طریقہ ہے۔ اگر آپ بعد میں لکھا گیا فنکشن کال کرنا چاہتے ہیں تو b.sum( ) لکھیں گے اور پہلے والا فنکشن کال کرنے کے لئے b.a::sum( ) لکھنا ہوگا۔

اسی طرح آپ اپنی ڈرائیوڈ اور بیس کلاس میں ایک ہی نام کے ڈیٹا ممبرز بھی ڈیکلیر کر سکتے ہیں اور ان کو ایکسیس کرنے کا طریقہ بھی اوپر کی طرح ہے اگر آپ ایک ڈیٹا ممبر temp کے نام سے دونوں کلاسز میں ڈیکلیر کرتے ہیں تو b.temp سٹیٹ مینٹ ڈرائیوڈ کلاس کو ایکسیس کرتی ہے جبکہ

b.a::temp آپ کی بیس کلاس کے ڈیٹا ممبر کو ایکسیس کرنے کے لئے استعمال ہوگی اور اس کو ڈومینیٹ (dominate) کہتے ہیں۔ آیئے اس کے لئے ایک پروگرام لکھتے ہیں۔

مثال نمبر 7.4 اووررائیڈنگ کلاس ممبر فنکشنز

```
class age
    {
    protected:
    int a;
    public:
    void setage(int);
    void show( );
    };
class derived:public age
    {
    public:
    void show( );                       //overriden function
    };
void age::setage(int v)
    {
    a=v;
    }
void age::show( )
    {
    cout <<"\n Base class member function:";
    cout <<"\n Your age:" <<a;
    }
void derived::show( )
    {
    cout <<"\n Derived class member function:";
    cout <<"\n Your age:" <<a;
void main(void)
    {
    clrscr( );
```

```
age a;
derived d;
a.setage(19);
d.setage(25);
a.show( );
d.show( );
d.age::show( );
getch( );
}
```

اس پروگرام میں ہم نے ایک کلاس age کے نام سے بنائی ہے اور ایک کلاس ڈرائیوڈ کے نام سے پہلی کلاس age سے ڈرائیو کی گئی ہے اور ان دونوں کلاسز میں ( )show کا فنکشن لکھا ہے۔ یعنی کہ ڈرائیوڈ کلاس میں یہ فنکشن اوورائیڈ کیا گیا ہے اور بعد میں ( )main میں اسے کال کیا گیا ہے۔ ()main میں دونوں کلاسز کے اوبجیکٹ بنائے گئے ہیں۔ اور ( )d.age::show پر غور کریں اس سے age کلاس کا ممبر فنکشن کال ہو رہا ہے لیکن ریفرنس ڈرائیوڈ کلاس کا ہے۔ اس پروگرام کی آؤٹ پٹ کچھ یوں ہوگی۔

```
Base class member function:
Your age : 19
Derived class member function:
Your age : 25
Base class member function:
Your age : 25
```

کنسٹرکٹر اور ڈسٹرکٹر: (Constructor and Destructor):

آپ اس بات سے بخوبی واقف ہیں کہ ایک ڈرائیوڈ کلاس اپنی بیس کلاس کے تمام ممبرز انہریٹ کرتی ہے اور جب ڈرائیوڈ کلاس کا اوبجیکٹ بنایا جاتا ہے تو بیس کلاس کا کنسٹرکٹر کال ہونا چاہئے جو بیس کلاس کے ممبرز کو اپنی شلائز کرے لیکن بیس کلاس کے کنسٹرکٹر اور آسائنمنٹ آپریٹرز ڈرائیوڈ کلاس انہریٹ نہیں کرتے۔ ہاں البتہ آپ کی ڈرائیوڈ کلاس کا کنسٹرکٹر اور آسائنمنٹ آپریٹرز بیس کلاس کے آسائنمنٹ آپریٹرز اور کنسٹرکٹرز کال کر سکتے ہیں۔ ڈرائیوڈ کلاس کا کنسٹرکٹر ہمیشہ بیس کلاس کا کنسٹرکٹر کال کرتا ہے۔ اگر آپ ڈرائیوڈ کلاس کا کنسٹرکٹر نہیں لکھتے تو ڈرائیوڈ کلاس کا ڈیفالٹ کنسٹرکٹر بیس کلاس کا ڈیفالٹ کنسٹرکٹر کال کرے گا لیکن ڈسٹرکٹر کال کرنے کا طریقہ اس کے الٹ ہے یعنی کہ ڈرائیوڈ کلاس کا ڈسٹرکٹر اس کی بیس کلاس کے ڈسٹرکٹر سے پہلے کال کیا جائے گا۔ اس کے لئے ہم نے ایک چھوٹا سا پروگرام نیچے لکھا ہے اس پر غور کریں۔

مثال نمبر 7.5 کنسٹرکٹر اور ڈسٹرکٹر کالنگ

```
class base
    {
```

```
public:
base( ):temp(4);                         //constructor
{
cout <<"\n Base class constructor:" <<temp;
~base( )
{                                        //destructor
cout <<"\n Base class desctructor:" <<temp;
}
protected:
int temp;
};
class derived:public base
{
derived( )                               //constructor
{
temp++;
cout <<"\n Derived class destructor:" <<temp <<endl;
}
~derived( )
{                                        //destructor
temp=0;
cout <<"derived class destructor:" <<temp;
}
};
void main(void)
clrscr( );
derived d;
getch( );
}
```

اس پروگرام میں ڈرائیوڈ کلاس بیس کلاس کے تمام ممبرز انہریٹ کر رہی ہے اور دونوں کلاسز کے ہم نے کنسٹرکٹر اور ڈسٹرکٹر لکھے ہیں اور main( ) میتھڈ میں ڈرائیوڈ کلاس کا اوبجیکٹ بنایا ہے۔ اس پروگرام کی آؤٹ پٹ کچھ یوں ہوگی۔

```
Base class constructor : 4
```

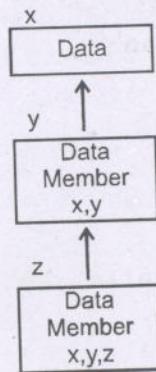
```
Derived class constructor : 5
Derived class destructor : 0
Base class destructor : 0
```

اس آوٹ پٹ پر غور کریں کہ بیس کلاس کا ڈسٹرکٹر ڈرائیوڈ کلاس کے ڈسٹرکٹر کے بعد پرنٹ ہوا ہے یعنی ہر بیس کلاس کا کنسٹرکٹر اپنی ڈائیوڈ کلاس سے پہلے ایگزیکیوٹ ہوتا ہے۔ جبکہ ڈسٹرکٹر اس کے الٹ ہے۔ ڈسٹرکٹر ہمیشہ پہلے ڈرائیوڈ کلاس کا کال ہوگا۔

ملٹی پل کلاسز:

اس سے پہلے آپ صرف ایک ڈرائیوڈ کلاس بناتے تھے جو بیس کلاس کی تمام خصوصیات ایکسیس کرسکتی ہے۔ آپ ایک سے زیادہ کلاسز بھی بنا سکتے ہیں۔ آیئے اس کے لئے ایک مثال دیکھتے ہیں۔

مثال نمبر 7.6 ملٹی پل کلاسز

```
class x
    {
    x( )
    cout <<"x class constructor:";}
    }
    };
class y:public x
    {
    y( )
    {cout <<\n y class constructor:";}
    }
class z:public x
    {
    Z( )
    {cout <<"\n z class constructor:";}
    };
void main(void)
    clrscr( );
    z z1;
    getch( );
    }
```

اس طرح کے پروگرام کی حرآرکی کچھ یوں ہوتی ہے۔

یعنی ہماری دوسری کلاس y میں اپنے اور x کے تمام ڈیٹا ممبرز شامل ہیں جن کی x اجازت دیتا ہے۔ اور z کلاس میں x اور y دونوں کی خصوصیات شامل ہیں۔

ورچوئل فنکشن:

ورچوئل کا مطلب ہے کہ ایک چیز کا ظاہر ہونا لیکن حقیقت میں وہ چیز نہیں ہوتی۔ اسی طرح C++ میں بھی ورچوئل فنکشن کچھ اس طرح کام کرتا ہے جب آپ ورچوئل فنکشن بناتے ہیں اور ایک پروگرام میں اس کلاس کا ورچویل فنکشن کال کرتے ہیں تو ہوسکتا ہے کہ حقیقت میں کسی مختلف (دوسری) کلاس کا فنکشن کال ہو رہا ہو۔ آپ ایک ورچوئل فنکشن کو بیس کلاس میں ڈیکلیر کر سکتے ہیں۔ اور پھر اسے کسی بھی ڈرائیوڈ کلاس میں ریڈیفائن کر لیں اور ایسے فنکشنز کی ٹائپ اور آرگومنٹ ایک جیسے بھی ہو سکتے ہیں۔ اب یہ ڈرائیوڈ کلاس میں دوبارہ ڈیفائن کیا گیا فنکشن بیس کلاس میں لکھے ہوئے اس نام کے فنکشن کو اوورائیڈ کردے گا۔ ورچوئل فنکشنز صرف ممبر فنکشن ہی ہو سکتے ہیں۔

آپ سوچ رہے ہوں گے کہ ورچوئل فنکشنز کی ضرورت کیوں پیش آتی ہے۔ تو فرض کریں کہ آپ کے پاس مختلف کلاسز کے کئی اوبجیکٹس ہیں لیکن آپ ایک ہی فنکشن کو کال کرتے ہوئے ان سب پر ایک ہی آپریشن پرفارم کرنا چاہتے ہیں تو ایسے کیس میں آپ کو ان فنکشنز کی ضرورت پیش آتی ہے۔ ایک بات اور ذہن میں رکھیں کہ اگر آپ کے پاس دو فنکشنز ایک ہی نام کے ہیں لیکن ان کے آرگومنٹس مختلف ہیں تو C++ انہیں مختلف فنکشن مانتی ہے اور وہ ورچوئل فنکشن کے طور پر کام نہیں کریں گے۔

یہ C++ کا بہت پاورفل فیچر ہے اور اس کو پولی مارفزیم بھی کہتے ہیں۔ یہ ورچوئل فنکشنز کی مدد سے پرفام کی جاتی ہے اس کو آپ بعد میں پڑھیں گے۔ فی الوقت آیئے ورچوئل فنکشن کو استعمال کرتے ہوئے ایک پروگرام بناتے ہیں۔

<u>مثال نمبر 7.7 سادہ ورچوئل فنکشن پروگرام</u>

```
class base
  {
  public:
  void display( )
  {
  cout <<"\n Base class virtual function:";
```

```
    }
    };
class derive 1:public base
    {
    public:
    void display( )
    {
    cout <<"\n Derived1 calls, derive::display( ):";
    }
    };
    class derive2:public base
    {
    public:
    void display
    {
    cout <<"\n derive2 class, derive2::display( ):";
    }
    };
void main(void)
    {
    clrscr( ); base b;
    derive1 d1;                  //derive1 class object
    derive2 d2;                  //derive2 class object
    base* p=&b;                  //pointer to base class
    p→display( );                //execute display
    p=&d1;                       //put address of d1 in p
    p→display( );                //execute display( )
    p=&d2;
    p→display( );
    getch( );
    }
```

ڈرائیوڈ 1 اور ڈرائیوڈ 2 کلاسز بیس کلاس سے ڈرائیو کی گئی ہیں اوران دونوں کلاسز میں ایک فنکشن ( )display کا ہے۔( )main میتھڈ میں ہم

نے ان دونوں کلاسز کے اوبجیکٹس d1 اور d2 بنائے ہیں اور بیس کلاس کو پوائنٹر آسائن کیا ہے۔اس کے بعد ہم نے اس پوائنٹر p کی مدد سے display ( ) کو کال کیا ہے۔اس کے بعد ڈرائیوڈ d1 کا ایڈریس اس پوائنٹر p میں سٹور کر دیا ہے اور p کے ریفرنس display( ) کو کال کیا ہے۔ d2 کے ساتھ بھی ایسا ہی کیا ہے۔ جب آپ اس پروگرام کو ایگزیکیوٹ کریں گے تو یہ آؤٹ پٹ ہوگی۔

```
Base class virtual function
Base class virtual function
Base class virtual function
```

آپ نے دیکھا کہ تمام کالز ایک ہی فنکشن کو جا رہی ہیں۔اس نے بیس کلاس کے فنکشن کے علاوہ کالز کو نظر انداز کر دیا ہے کیونکہ بیس کلاس کے لئے ہم نے پوائنٹر ڈیکلیر کیا ہے۔

آپ تمام کوڈ یہی رہنے دیں صرف ایک لفظ کا اضافہ کر دیں اور پھر آؤٹ پٹ دیکھیں۔

```
class base
    {
    public:
    virtual void display( )
    {
    cout <<"\n Base class pure virtual function:";
    {
```

اب دوبارہ اس پروگرام کو ایگزیکیوٹ کریں اس کی آؤٹ پٹ تبدیل ہو چکی ہو گی اب اس کی آؤٹ پٹ یہ ہو گی۔

```
Base class virtual function
Derive1 class, Derive1::display( );
Derive2 class, Derive2::display( );
```

اس کو پولی مارفزیم بھی کہتے ہیں۔ایک فنکشن کو کال p→display( ) کیا جا رہا ہے لیکن حقیقت مختلف کلاسز کے مختلف فنکشنز ایگزیکیوٹ ہو رہے ہیں آپ نے پہلے جو پروگرام لکھا تھا وہ سادہ ورچوئل فنکشن کی ایک مثال تھی لیکن ایک لفظ ورچوئل لکھنے سے پروگرام کا تمام اصول تبدیل ہو گیا۔ یہ ورچوئل فنکشن کی ایک مثال ہے۔اس کے علاوہ آپ پہلے پروگرام میں یہ دو لائنز بے شک نہ لکھیں۔

```
base b;
base* p=&b;
```

لیکن دوسری دفعہ جب ورچوئل کی ورڈ لکھیں گے تو یہ لکھنا ہوگا۔

Abstract کلاسز:

فرض کریں آپ نے بیس کلاس میں ایک فنکشن ڈیکلیر کیا ہے اور بعد میں آپ اس کلاس سے ڈرائیوڈ ہونے والی تمام کلاسز میں اس فنکشن کو اوورائیڈ کرنا چاہتے ہیں تو پھر آپ اس ورچوئل فنکشن کو pure ورچوئل فنکشن بنا لیں یعنی بیس کلاس میں اس کی باڈی لکھنے کی ضرورت نہیں ہوتی یا دوسرے الفاظ میں اس کی باڈی بیس کلاس میں نہیں لکھی جائے گی۔ یوں ایک pure ورچوئل فنکشن ایسا فنکشن ہوتا ہے جس کی اپنی کلاس (بیس) میں کوئی باڈی نہیں

ہوتی۔اس کا جنرل طریقہ یہ ہے۔

```
virtual void x( )=0;
```

تو یوں abstract کلاس ایسی کلاس ہوتی ہے جس میں کم از کم pure ورچوئل ممبر فنکشن موجود ہو اور ایسی ڈرائیوڈ کلاس جس میں pure ورچوئل فنکشن نہیں ہوگا وہ کنکریٹ (concrete) کلاس کہلاتی ہے۔

نوٹ: ہم نے آپ کو ایک بات پہلے بھی بتا چکے ہیں کہ ہر ورچوئل فنکشن ممبر فنکشن ہوتا ہے تو ہر کلاس کی ڈرائیوڈ کنکریٹ کلاس کو ورچوئل ممبر فنکشن کی باڈی (ڈیفینیشن) لازمی لکھنا ہوتی ہے۔اس کا ہم نے ایک سادہ سا پروگرام نیچے لکھا ہے تا کہ آپ کو سمجھنے میں آسانی رہے۔

مثال نمبر 7.8 Abstract کلاس کی مثال

```
class base{
    public:
    virtual void change(int)=0;          //virtual function
    virtual void display()=0;
    protected:
    int i,c;
    };
class derive1:public base
    {
    public:
    void change(int d)
    {
    i=d;
    }
void display( )
    {
    cout <<"\n Derive1.display( ):" <<i;
    }
    };
class derive2:public base
    {
    public:
    void change(int d)
    {
    i=d;
    i--;
```

```
    }
void display( )
    {
    cout <<"\n derive2.display( ):" <<i
    }
    };
void main(void)
    {
    clrscr( );
    derive d1;
    derive d2;
    d1.change(7);
    d1.display( );
    d2.change(10);
    d2.display( );
    getch( );
    }
```

اس پروگرام میں ایک Abstract کلاس بنائی گئی ہے جس کا نام base ہے اور اس کے دو ممبر فنکشنز ورچوئل ڈیکلیر کئے گئے ہیں۔ اس کے بعد دو مزید کلاسز بنائی ہیں جو اس base کلاس سے ڈرائیوڈ کی گئی ہیں۔ ان میں ان فنکشنز کی ڈیفینیشن تحریر کی گئی ہے۔ جب آپ اس پروگرام کو ایگزیکیوٹ کریں گے تو اس کی آؤٹ پٹ یہ ہوگی۔

```
Derive1.display( ) : 7
Derive2.display( ) : 10
```

ورچوئل ڈسٹرکٹر: (Virtual Destructor):

آپ نے اس سے پہلے ورچوئل فنکشنز کے بارے میں پڑھا ہے کہ یہ ایسے فنکشن ہوتے ہیں جنہیں بیس کلاس میں ورچوئل ڈیکلیر کیا جاتا ہے اور سب کلاس میں یہ فنکشنز اوورائیڈ کئے جاتے ہیں۔ اس میں فنکشن کا نام اور پیرامیٹرز ایک جیسے ہوتے ہیں۔ اس سے ظاہر ہوتا ہے کہ کنسٹرکٹر اور ڈسٹرکٹر ورچوئل ڈیکلیر نہیں کئے جاسکتے۔ یہ بات کنسٹرکٹر کے لئے درست ہے لیکن ڈسٹرکٹر کا میتھڈ اس کو غلط قرار دیتا ہے۔ ہر کلاس کا ایک منفرد ڈسٹرکٹر ہوتا ہے اور آپ اسے ورچوئل بھی ڈیکلیر کر سکتے ہیں۔ آیئے اس کی ایک مثال دیکھتے ہیں۔

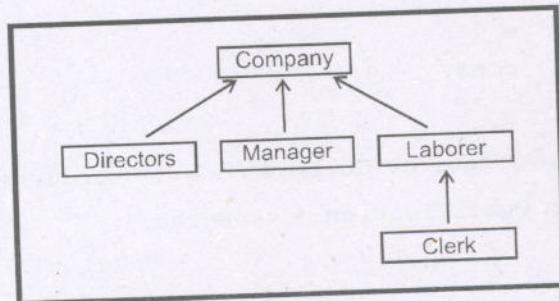
<u>مثال نمبر 7.9 ورچوئل ڈسٹرکٹر</u>

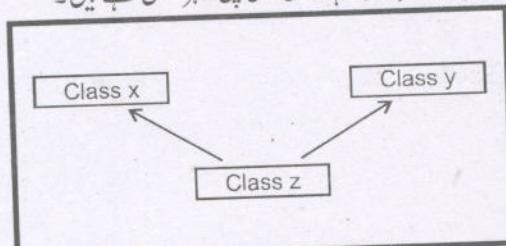
```
class super
    {
    public:
    //~super( )
```

```
virtual ~super( );
{
cout <<"\n Super destructor destroyed:";
}
};
class derived:public super
{
public:
derived( )
{
cout <<"\n Derived destructor destroyed:";
}
};
void main(void)
cout <<"\n Enter your information:";
p1.getname( );
cout "\n ////////////";
cout <<"\n Employee information:";
emp.showname( );
cout <<"\n ///////////";
cout <<"\n Your data";
p1.showname( );
getch( );   }
```

یہ ملٹی پل انہریٹینس کی سادہ سی مثال ہے لیکن مثال خواہ سادہ ہو یا مشکل، ملٹی پل انہریٹینس کا طریقہ یہی ہے۔اس طرح آپ ایک کلاس کو کئی کلاسز سے بھی ڈرائیو کر سکتے ہیں اور یہ ڈرائیوڈ کلاس اپنی تمام بیس کلاسز کے ممبر انہریٹ کر سکتی ہے۔

انہریٹینس کے لیولز:

آپ نے اوپر ابھی تک صرف سادہ انہریٹینس پڑھی ہے۔اس کے علاوہ بھی انہریٹینس کے کئی طریقے ہیں مثلاً آپ ایسی کلاسز بھی بنا سکتے ہیں جو ایسی کلاس سے ڈرائیوڈ کی گئی ہو جو مزید کسی کلاس کی سب کلاس ہے۔یعنی آپ کی بیس کلاس کسی اور کلاس سے ڈرائیوڈ کی گئی ہے۔مثلاً

```
class x{ };
class y:public x{
//rest of code
class z:public y{
```

```
//code here
}
```

اب یہاں پر y کلاس x سے ڈرائیوڈ کی گئی ہے اور z کلاس y سے ڈرائیوڈ کی گئی ہے۔ آپ اس کو حرآرکی کی مدد سے سمجھنے کی کوشش کریں کہ یہ کلاسز کس طرح کام کرتی ہیں۔

ملٹی پل انہریٹینس: (Multiple Inheritance):

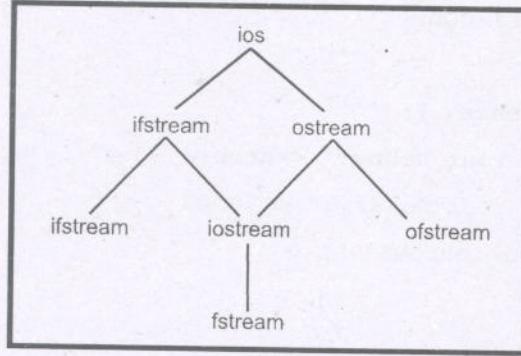
ایک کلاس ایک سے زائد کلاسز سے بھی ڈرائیو کی جاسکتی ہے۔ اس کو ملٹی پل انہریٹینس کہتے ہیں۔

اس چارٹ سے ظاہر ہوتا ہے کہ کلاس z دو کلاسز x,y سے ڈرائیوڈ کی گئی ہے لہٰذا اس میں دونوں کلاسز کی خصوصیات شامل ہیں۔ اس سے ظاہر ہوا کہ ملٹی پل انہریٹینس میں ایک ڈرائیوڈ کلاس کئی بیس کلاسز کے ممبرز کو انہریٹ کرتی ہے۔ یہ C++ کا بہت پاورفل فیچر ہے۔ آئیے ملٹی پل انہریٹینس کے لئے ایک پروگرام لکھتے ہیں۔

مثال نمبر 7.10 ملٹی پل انہریٹینس

```
include<stdio.h>
coust int size=45;                    //constant variable
class information
    {
    private:
    char College[size];
    char Degree[size];
    public:
  void input( )
     {
     cout <<"Enter College Name:";
```

```
gets(college);
cout <<"\n Enter Qualification:"
<<"\n (MBA, MCS, BSc, B.A):";
cin >>degree;
}
void output( ) const
{
cout <<"\n School or College name:" <<college;
cout <<"\n Qualification:" <<degree;
}
};
class name
{
private:
char name[size];
int age;
public:
void getname( )
{
cout <<"\n Enter name:";
gets(name);
cout <<"\n Enter age:";
cin >>age;
}
void showname( ) const
{
cout <<"\n Your name:" <<name;
cout <<"\n Your age:" <<age;
}
};
class employee:private name, private information
{
private:
```

```
    char hobb[size];
    int temp;
void getname( )
    {
    Name::getname( );
    cout <<"\n Enter your hobby:";
    gets(hobb);
    cout <<"\n Expected salary:";
    cin >>temp;
    intormation::input( );
    }
void showname( ) const
    {
    Name::showname( );
    cout <<"\n Your hobby:" <<hobb;
    cout <<"\n Your salary:" <<temp;
    information::output( );
    }
    };
class personal:public Name
    {
    };
void main(void)
    {
    clrscr( );
    employee emp;
    personal p1;
    cout <<"\n Enter Data for Employee1:";
    cmp.getname( );
    cout <<"\n //////////";
    getch( );
    }
```

اس پروگرام میں ایک کلاس person کے نام سے بنائی ہے جس کے کچھ ممبر فنکشن اور ڈیٹا ممبرز ہیں۔اس پروگرام کے ( )main فنکشن میں ہم

نے fstream کا اوبجیکٹ pfile بنایا ہے اور اس کے کنسٹرکٹر کو دو آرگومنٹس پاس کئے ہیں۔ ایک فائل کا نام اور دوسرا فائل موڈ، ہم نے فائل موڈ دو تحریر کئے ہیں۔ ios::in اور ios::app یعنی یہ فائل ان پٹ اور آؤٹ پٹ دونوں کے لئے ہیں۔ ios::app کا یہ فائل موڈ فائل میں پہلے سے موجود ریکارڈ کو ڈسٹرب نہیں کرتا اور نئے ریکارڈز فائلز کے آخر پر لکھتا ہے۔ اس کے علاوہ ہم نے یہ دیکھا ہے کہ فائل میں کل کتنے ریکارڈز ہیں اور اگر آپ کوئی ریکارڈ تلاش کرنا چاہتے ہیں تو وہ بھی کر سکتے ہیں۔

فائل پروسیسنگ:

C++ میں فائل پروسیسنگ کے لئے آپ کو <iostream.h> اور <fstream.h> ہیڈر فائلز شامل کرنا ہوتی ہیں۔ C++ اصل میں تمام فائلز کو بائٹس کی ایک ترتیب تصور کرتی ہے اور ہر فائل کے اختتام پر اینڈ آف فائل کا مارک لگا ہوتا ہے یا پھر آخری ریکارڈ کا نمبر ہوتا ہے۔ جب ایک فائل اوپن کی جاتی ہے تو اس کا اوبجیکٹ بنتا ہے اور سٹریم جو اس اوبجیکٹ سے منسلک ہوتی ہے وہ بھی اوپن ہو جاتی ہے۔ C++ فائلز پر کوئی سٹرکچر عائد نہیں کرتی۔ اس لئے پروگرامرز کو اس میں خود سے اپنی ایپلی کیشن پر سٹرکچر لاگو کرنا ہوتا ہے۔ ان تمام کاموں کے لئے C++ کی io کلاس استعمال کی جاتی ہے۔ اس کی حرارکی کچھ یوں ہے۔

I/O کی حرارکی

آپ کسی بھی فائل میں یہ ios کلاس کس طرح استعمال کر سکتے ہیں۔ آیئے اس پروگرام میں دیکھتے ہیں۔

مثال نمبر 7.11 ایک Sequential فائل بنانا

```
#include<iostream.h>
#include<fstream.h>
#include<stdlib.h>
#include<conio.h>
void main(void)
    {
    clrscr( );
    ofstream student("c://lib.dat, ios::out);
    if(!student)
    {                          //overload!operator
```

```
cerr("\n File could not be opened");
exit(1);
}
cout <<"\n Enter name, class, rollno:"
<< "\n For end press F6 or CTRL+Z:\n";
char name[20], class1[7];
int rno;
while (cin >>name >>class1 >>rno)
{
student <<name' ' <<class1 <<' ' <<rno;
}
getch( );
}
```

اب اس پروگرام کو ہم ایک نظر دیکھتے ہیں۔اس میں ایک فائل اوپن کی گئی ہے اور فائل آؤٹ پٹ کے لئے کھولی گئی ہے۔اس لئے ofstream اوبجیکٹ کال کیا گیا ہے۔اس اوبجیکٹ کے کنسٹرکٹر کو دو آرگومنٹس پاس کئے گئے ہیں۔ایک فائل کا نام اور دوسرا فائل کا موڈ، کہ وہ کس موڈ میں اوپن کی گئی ہے۔یہ موڈ کیا ہے آپ آگے پڑھیں گے۔اگر یہ فائل اوپن ہو جاتی ہے تو ٹھیک ورنہ if کنڈیشن ایگزیکیوٹ ہوگی۔

```
if(!student)
```

اس کنڈیشن میں ہم نے cerr استعمال کیا ہے یہ ایرر میسج آؤٹ پٹ کے لئے استعمال ہوتا ہے۔اگر فائل اوپن نہیں ہوگی تو if کنڈیشن میں لکھا ہوا میسج ڈسپلے ہوگا اور پھر پروگرام ٹرمینیٹ کر دیا جائے گا۔ہم نے ان پٹ while لوپ میں لی ہے اور یہ اس وقت تک ان پٹ لیتا رہے گا جب تک کہ آپ F6 یا CTRL+Z پریس نہیں کرتے۔

کسی بھی فائل کو اوپن کرنے کے لئے آپ مندرجہ ذیل موڈ میں سے کسی بھی موڈ کا انتخاب کر سکتے ہیں۔کب کون سا موڈ استعمال کرنا ہے۔آیئے دیکھتے ہیں۔

موڈ	وضاحت
ios::app	یہ تمام آؤٹ پٹ فائل کے آخر پر لکھتا ہے
ios::ate	اس میں ڈیٹا فائل میں کہیں بھی لکھا جا سکتا ہے یہ فائل کو آؤٹ پٹ کے لئے اوپن کرتا ہے اور ڈیفالٹ کرسر فائل کے آخر پر ہوتا ہے
ios::out	یہ آؤٹ پٹ کے لئے فائل اوپن کرتا ہے
ios::in	یہ فائل ان پٹ کے لئے اوپن کرتا ہے
ios::nocreate	اگر فائل پہلے نہیں ہوگی تو اوپن آپریشن ناکام ہو جائے گا
ios::noreplace	یہ فائل میں تبدیلی نہیں کرنے دیتا یعنی اگر فائل پہلے سے موجود ہوگی تو یہ اوپن نہیں کرے گا

آپ نے جو مثال پہلے دیکھی ہے وہ بہت سادہ مثال ہے۔آیئے سیکوینشل فائل بنانے کے لئے ایک اور پروگرام لکھتے ہیں۔

مثال نمبر 7.12 آؤٹ پٹ اور ان پٹ فائل

```
#include<iostream.h>
#include<conio.h>
#include<fstream.h>
#include<stdio.h>
class person
  {
  private:
  char name[20];
  int age;
  public:
void showdata( )
  {
  cout <<"\n Name:" <<name;
  cout <<"\n Age:" <<age;
  }
void read( )
  {
  cout <<"\n Enter Name:";
  gets(name);
  cout <<"\n Enter Age:";
  cin >>age;
  }
  };
void main( )
  {
  clrscr( );
  person p1;
  char ch;
  int no;
  fstream pfile;
  pfile.open(c:\\p1.dat", ios::app|ios::in);
  do
```

```
{
cout <<Enter Data:";
p1.read( );
pfile.write((char*)&p1. sizeof(person));
cout <<"\n Do you want to continue:";
cin >>ch;
}
while(ch=='y'||ch=='y');
pfile.seekg(0);
pfile.seekg(0, ios::end);
int space=pfile.telly( );
int trecord=space/sizeof(person);
cout <<"\n There are" <<trecords <<"records in library";
cout <<"\n Enter no to read:";
cin >>no;
int position=(no-1)*sizeof(person);
pfile.seekg(position);
pfile.read((char*)&p1, sizeof(person));
p1.showdata( );
{
clrscr( );
super* ptr=new derived;
delete ptr;
getch( );
}
```

اس پروگرام کو ایگزیکیوٹ کریں گے تو اس کی آؤٹ پٹ یہ ہوگی۔

```
Derived destructor destroyed :
Super destructor destroyed :
```

آپ نے دیکھا کہ دونوں کلاسز کے ڈسٹرکٹر ختم ہو گئے ہیں۔ اس میں ہم نے ایک لائن کو کومنٹس میں لکھا ہے۔

```
//~Super( );
```

آپ اس کے کومنٹس ختم کر دیں اور اس سے نیچے والے ورچوئل ڈسٹرکٹر کو کومنٹس دیں اور پھر آؤٹ پٹ دیکھیں۔ ایسا کرنے سے صرف ایک ڈسٹرکٹر ختم ہوگا۔ اس کی آؤٹ پٹ یہ ہوگی۔

```
Super destructor destroyed :
```

سوال نمبر 1: ان سوالات کے مختصر جواب دیں۔

(i) پولی مارفزیم کیا ہے؟

(ii) ایک Abstract بیس کلاس کیا ہوتی ہے؟

(iii) کمپوزیشن اور انہریٹینس میں کیا فرق ہوتا ہے؟

(iv) ورچوئل اور پیور (Pure) ورچوئل ممبر فنکشن کیا ہوتے ہیں؟

سوال نمبر 2: ایک پروگرام تحریر کریں جس میں دو کلاسز بنائی گئی ہوں اور تیسری کلاس ان دونوں کلاسز سے ڈرائیوڈ کی گئی ہو۔اس پروگرام میں ایک کتاب سے متعلقہ تفصیل ہو یعنی کہ اس کے پیجز اور قیمت وغیرہ کتنی ہے؟

جوابات

❶: جواب

(i) پولی مارفزیم میں آپ ایک سب کلاس اور اس کے میتھڈز کسی بھی کلاس کی حرآرکی میں شامل کر سکتے ہیں اور یہ اس کلاس کے انٹرفیس کو ڈسٹرب نہیں کرتا یعنی اس کو تبدیل کرنے کی ضرورت نہیں ہوتی۔

(ii) Abstract بیس کلاس ایسی بیس کلاس ہوتی ہے جس میں کم از کم ایک پیور (Pure) ورچوئل فنکشن شامل ہو اور اس کلاس کا آپ اوبجیکٹ نہیں بنا سکتے۔

(iii) کلاسز کی کمپوزیشن میں ایک کلاس کو دوسری کلاس کا ممبر ڈیکلیر کیا جاتا ہے جبکہ انہریٹینس میں آپ ایک کلاس سے دوسری کلاس ڈرائیو کرتے ہیں۔

(iv) ورچوئل ممبر فنکشن ایسا فنکشن ہوتا ہے جو کہ سب کلاس میں اوورائیڈ کیا جا سکتا ہے۔ جبکہ پیور ورچوئل فنکشن ڈائریکٹ کال نہیں کیا جا سکتا۔ ڈرائیوڈ کلاس میں صرف اس کے اوورائیڈ کیئے ہوئے فنکشن کال کیئے جا سکتے ہیں۔ اور یہ صفر (0) سے اینی شلائز کیا جاتا ہے۔

❷: جواب

```
const int month=5;
class JBD  {
    private:
    char title[25];
    int cost;
    public:
void input( )  {
    cout <<"\n Enter title:";
    gets(title);
    cout <<"\n Enter price:";
    cin >>cost;
    }
void display( )    {
    cout <<"\n Title:" <<title;
    cout <<"\n Price:" <<cost;
    }
    };
class shop     {
    private:
    float sarr[month];
    public:
    void input( );
    void display( );   {
    cout <<"\n Enter sale for 5 months:\n";
    for(int i=8; i<month; i++)     {
    cout <<"month" <<i+1 <<":";
```

```
        cin >>sarr[i];
        }
        }
    void shop::display( )
        {
        for(int i=8; i<months; i++)
        cout <<"\n Sales for month:" <<i+1 <<":")
        cout <<sarr[i];
        }
        }
    class cbook:private JBD, private shop
        {
        private:
        int page;
        public:
        void input( )
        {
        JBD::input( );
        cout <<"\n Enter number of pages:";
        cin >>page;
        shop::input( );
        }
        void display( )
        {
        JBD::display( );
        cout <<"\n Total pages are:" <<page;
        shop::display( );
        }
        },
    void main
        {
        clrscr( )
        cbook c1;
        c1.input( );
        c1.display( );
        getch( );
        }
```